A STABILITY ANALYSIS OF SPARSE K–MEANS

by

Abraham Apfel

BA, Ner Israel Rabbinical College, 2012

MA, Ner Israel Rabbinical College, 2014

Submitted to the Graduate Faculty of

the Graduate School of Public Health in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

GRADUATE SCHOOL OF PUBLIC HEALTH

This dissertation was presented

by

Abraham Apfel

It was defended on

May 5th 2017

and approved by

George Tseng, PhD, Professor, Department of Biostatistics, Graduate School of Public Health, University of Pittsburgh

Yan Lin, PhD, Research Associate Professor, Graduate School of Public Health, Department of Biostatistics, University of Pittsburgh

Dana Tudorascu, PhD, Assistant Professor, Department of Medicine, Psychiatry and Clinical Translational Science, School of Medicine, and Department of Biostatistics, Graduate School of Public Health, University of Pittsburgh

Dissertation Director: Stewart Anderson, PhD, Professor, Department of Biostatistics, Graduate School of Public Health, University of Pittsburgh

Copyright © by Abraham Apfel
2017

A STABILITY ANALYSIS OF SPARSE K–MEANS

Abraham Apfel, PhD

University of Pittsburgh, 2017

ABSTRACT

Sparse K-Means clustering is an established method of simultaneously excluding uninformative features and clustering the observations. This is particularly useful in a high dimensional setting such as micro-array. However the subsets of features selected is often inaccurate when there are overlapping clusters, which adversely affects the clustering results. The current method also tends to be inconsistent, yielding high variability in the number of features selected.

We propose to combine a stability analysis with Sparse K-Means via performing Sparse K-Means on subsamples of the original data to yield accurate and consistent feature selection. After reducing the dimensions to an accurate, small subset of features, the standard K-Means clustering procedure is performed to yield accurate clustering results. Our method demonstrates improvement in accuracy and reduction in variability providing consistent feature selection as well as a reduction in the clustering error rate (CER) from the previously established Sparse K-Means clustering methodology. Our method continues to perform well in situations with strong cluster overlap where the previous methods were unsuccessful.

Public health significance: Clustering analysis on transcriptomic data has shown success in disease phenotyping and subgroup discovery. However, with current methodology, there is a lack of confidence in terms of the accuracy and reliability of the results, as they can be highly variable. With our methodology, we hope to allow the researcher to use cluster analysis to achieve disease phenotyping and subgroup discovery with confidence that they are uncovering accurate and stable results thus ensuring that their findings will allow reliable public health decisions to be made from their work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 BACKGROUND

1.1 INTRODUCTION

Most standard statistical techniques used to analyze and summarize data rely on the luxury of having more observations than variables of interest. Over the last twenty five years, there has been an increasing interest in the analysis of datasets with many dimensions. This is largely due to the development of new technologies enabling us to discover and record detailed information in such fields as genetics and neuroimaging. With the continuous advances in computing power, the growth of these fields and many others like them produce a need to develop statistical methods to analyze their growing datasets properly. In particular, datasets with more variables than observations pose a unique challenge to the statistical world. For example, one PET scan of an individual's brain can contain hundreds of thousands of voxels. Each voxel can be viewed as a separate variable of interest. However, due to the high cost of imaging, a typical dataset may consist of only 60 – 100 individual scans. Thus, the number of variables, often labeled p, is much larger than the number of observations, often labeled n.

One of the major challenges with high dimensional data is the danger of overfitting the data. When there are more variables than observations, a simple least squares regression will fit the data perfectly, however this will lead to high variability in the results and an inability to predict accurately.

Various methodologies have been developed to analyze such data. One approach is a shrinkage-based approach, such as ridge regression [Hoerl and Kennard, 1970] or the least absolute shrinkage and selection operator (commonly referred to as "Lasso") [Tibshirani, 1996]. These approaches use least squares regression but add a penalty which shrinks the

coefficients values towards zero. Ridge regression uses an L_2 penalty, while the Lasso uses an L_1 penalty. This results in only the most important variables playing a significant role in the model even if initially, there were many to choose from. These penalties allow the model to perform well with a strong decrease in variance in exchange for only a small increase in bias even in situations where regular least squares regression fails, such as when $p \gg n$ [James et al., 2013].

Ridge regression and the Lasso each have advantages over each other. Ridge regression performs better in situations where the covariates are correlated with each other, whereas the Lasso does not perform well in that setting [Friedman et al., 2001]. However, the Lasso has a unique advantage in that it can force some of the coefficients to be exactly equal to zero and thus exclude them from the model. Thus sparsity can be achieved via the Lasso whereas by ridge regression, every coefficient must be included in the model which can make the interpretation challenging. More recently, other penalties have been developed to combine the advantages of both the L_1 and L_2 penalties, to achieve sparsity while also working with correlated data. Examples of such penalties are the elastic net [Zou and Hastie, 2005], Grouped Lasso [Yuan and Lin, 2006], fused Lasso [Tibshirani et al., 2005], and the OSCAR [Bondell and Reich, 2008].

Another approach to working with high dimensional data is in the form of dimension reduction. There are two widely used forms of dimension reduction, Principal Components Regression (PCR) and Partial Least Squares (PLS)[James et al., 2013]. The basic concept is that instead of running a model with p covariates, these covariates can instead be summarized by $m \leq p$ principal components, where each one of the m principal components is a linear combination of the p covariates. These principal components are then put in the model as the new covariates. Similar to ridge regression, the disadvantage of these dimension reduction methods is that every covariate is used in the model and thus, no feature selection is performed. In fact, it can be shown that ridge regression is a continuous form of PCR [Friedman et al., 2001].

The distinction between PCR and PLS is in the way that the principal components are calculated. In PCR, the principal components are calculated by choosing the direction in which the data vary the most. It is not based on the response variable at all. It is built on

the assumption that the direction in which the covariates show the most variation is also the direction which is most associated with the response variable. Thus there is no need to look at the response variable when calculating the principle components. This provides a useful technique in the unsupervised setting, as will be discussed below. PLS however, chooses the components based on their association with the response variable. This does not fit the predictors as closely as PCR does, but fits the response more closely. This can often lead to a decrease in bias but an increase in variance.

Although the above approaches are effective, they are only useful in a supervised learning setting. This means that for each observation of the predictor measurements there is an associated response variable. However, for an unsupervised setting, where we are lacking information as to the true underlying value of the responses we are trying to predict, we are still challenged.

There are two primary methodologies used to analyze unsupervised learning. One of them is Principal Components Analysis (PCA) the other is Clustering, each of them are useful in a high dimensional setting but have different accomplishments. PCA represents the high dimensional data in a low dimensional way, which can then be used to explain much of the variation amongst the data. Clustering looks to find homogeneous subgroups amongst the observations [James et al., 2013].

One can perform PCA [Dunteman, 1989] to summarize high dimensional data in a low dimensional fashion. After obtaining the $m \leq p$ principal components as described above, instead of then performing a regression on them as in a supervised setting, one can then plot the principal components against each other for a low dimensional view of the data.

Essential to any high dimensional dataset is the need to simultaneously cluster the data and select the key features responsible for distinguishing the clusters. Clustering analysis on transcriptomic data has shown success in disease phenotyping and subgroup discovery, which is a first step towards personalized medicine. The identified subtypes are clinically meaningful, with each subtype showing its distinct molecular pathway, treatment response and survival character. Success at identifying the underlying subtypes behind the transcriptomic data relies on unsupervised clustering algorithms. In the literature, various clustering algorithms are developed, including hierarchical clustering [Eisen et al., 1998], K-Means [Dudoit

and Fridlyand, 2002], mixture model approaches [Xie et al., 2008, McLachlan et al., 2002], resampling methods [Kim et al., 2009, Swift et al., 2004], tight clustering [Tseng and Wong, 2005], integrative analysis [Huo et al., 2016] amongst others.

There are many types of clustering methods. The two that are best known are K–means clustering and hierarchical clustering. Although both methodologies aim for a common result of grouping the observations so that each group contains the observations most similar to each other, their methodologies and assumptions are different and thus different scenarios warrant the use of different algorithms depending on what is known prior to the analysis.

For example, in scenarios where the data is skewed (e.g. log-normal distribution) or ellipsoidal (e.g. Normal Distribution with unequal variances), K–means can perform poorly [Steinley, 2006]. Another scenario where K–means has poor performance is if there is one large cluster with many small clusters and an alternative clustering mechanism, such as hierarchical clustering would be recommended. Nonetheless, generally speaking, K–means is considered the most influential algorithm for unsupervised learning [Steinley and Brusco, 2011].

There are two types of hierarchical clustering. The most commonly used method is agglomerative hierarchical clustering which is a bottom-up approach whereas the less commonly used method is divisive hierarchical clustering which is a top-down approach. The structure of agglomerative hierarchical clustering is that the two observations most similar to each other are grouped together. Following this, the next two closest observations are grouped together, etc. Branches may be formed linking individual observations as well as branches linking groups with other grouped or ungrouped observations. Eventually, at the top of the diagram, every observation is linked to one group. On the bottom, each individual observation is by itself. When complete, one can see a tree diagram showing various grouping of the observations and can determine how many clusters there should be based on this diagram.

Divisive hierarchical clustering works in a similar way but in the opposite direction, as initially all of the observations are viewed together as one cluster and then the longest distance between two data points or clusters is determined. The large initial cluster is continuously divided until each observation is by itself at the bottom of the tree.

In both of these methods, the number of clusters is decided upon only after the algorithm is run and will depend on where one "cuts" the resulting diagram. This can sometimes lead to an ambiguous result if it is not clear where to cut the diagram. One disadvantage of hierarchical clustering is that it is based on the assumption that any solution with a larger amount of clusters is nested within the solution with a smaller amount of clusters, which may not always be the case [Friedman et al., 2001]. For example, consider a dataset with three nationalities made of both males and females. If it was known a priori that there are only two clusters, logically the two clusters would be based on the gender of the subject. However, if there were three clusters it should be based on the nationality. Thus, hierarchical clustering may not always provide the most accurate separation of clusters [James et al., 2013].

1.2 K–MEANS CLUSTERING

In K–means clustering, one must first specify the number of clusters, K. There are also no overlapping clusters and every observation must belong to a cluster. Good clustering is accomplished when the within-cluster variation is minimized, which consequently maximizes the between-cluster variation. As described in Steinley and Brusco [2011], "objects within a cluster should be more similar than objects that are in different clusters". A thorough overview is provided in Jain [2010].

1.2.1 K–Means Clustering Algorithm

Let x_{ij} denote an observation in a dataset of size n by p, where there are n observations and p variables observed per observation, $i = 1, \ldots, n$ and $j = 1, \ldots, p$. K–means clustering attempts to solve

$$\min_{C_1,\ldots,C_k} \left\{ \sum_{k=1}^{K} \frac{1}{n_k} \sum_{i \in C_k} \sum_{j=1}^{p} (X_{ij} - \mu_{kj})^2 \right\} \tag{1.1}$$

where $C_1, \ldots, C_k$ are the clusters and n_k is the number of observations within each cluster. μ_{kj} is the mean for the j^{th} variable in the k^{th} cluster. This equation is very difficult to solve globally, instead the following algorithm is used to find a local minimum:

1. Randomly assign a number from 1 to K to each observation;
2. Compute the cluster centroid (the vector of p feature means for the observations in the k^{th} cluster);
3. Reassign each observation to the cluster whose centroid is closest (smallest in Euclidean distance); and
4. Repeat until cluster assignments stop changing.

By initializing the algorithm with random classifications for each observation, K random initial cluster centers will result. Then the Euclidean distance between each of the observations and each of the cluster centers is calculated and the observations are reassigned membership to the cluster for which there is the smallest distance between the observation and the cluster center. After each of the observations is assigned to a cluster, the cluster centers are then recalculated to reflect the mean value between all of the observations assigned to that cluster. This process is repeated until the observations no longer switch groupings and the cluster means are stable. The goal of K–means clustering is to minimize the within cluster distances which then also maximizes the between cluster distances. The distance is determined by summing up the Euclidean distance across all variables.

There is no guarantee however that this final grouping is the global minimum, as the initial cluster centers have an impact on the final result and often only a local minimum is achieved. In fact, Steinley [2006] asserts that even for data sets of moderate size, there can be thousands of local optima. Steinley [2008] advises to repeat the procedure several thousand times with different initial cluster centers and choose the best solution, the one with the minimum within cluster sum of squares.

One limitation of K–means clustering is that one must declare how many clusters one anticipates to have prior to the analysis. There have been many methodologies developed to determine how many clusters one should assume such as the gap statistic proposed by Tibshirani in 2001 among many others [Fang and Wang, 2012, Monti et al., 2003, Steinley and Brusco, 2011, Tibshirani and Walther, 2005, Zou and Hastie, 2005]. However, many of these methodologies are not accurate in a high dimensional setting.

Another limitation in K–means clustering is that every observation must be assigned to a cluster, even if in reality it is really an outlier. This resulting "noise" can potentially

distort the true clusters and make the results difficult to interpret. An area where this can be particularly problematic is in microarrays where every gene is assigned to a cluster, even though many genes are known to be unrelated to the biological process being investigated [Tseng and Wong, 2005].

A third problem with K–means clustering is that when using the K–means clustering algorithm to determine which cluster each observation belongs to, each variable is treated equally. If there are five variables being measured each one is weighted the same in determining the total Euclidean distance between the observation and its cluster mean. This is problematic in situations where the true underlying clusters are distinguished by some features more than others. This is particularly problematic in situations where there are many variables, especially when they outnumber the observations. These datasets can contain many uninformative variables. Additionally, as the number of dimensions increase, there is an increased likelihood of only achieving a local maximum. Much research has been performed to obtain a sparse clustering result in a high dimensional setting [Azizyan et al., 2014, Pan and Shen, 2007, Sun et al., 2012, Witten and Tibshirani, 2012].

As mentioned above, one problem with K–means clustering is that it automatically groups every observation into a cluster, which can distort the final results. Tseng and Wong [2005] developed a modified version of K–means clustering known as Tight Clustering, which removes the noisy observations from the analysis. This is done by using resampling to distinguish true cluster membership from those which are just by chance. It is assumed that when resampled, the noisy observations will vary which cluster they belong to whereas the observations with true cluster membership will consistently belong to the same cluster. Co-membership matrices are used for cluster identity. The tightness of the clusters is controlled by a tuning parameter and the final number of clusters is determined when the Tight Clustering algorithm no longer produces tight clusters.

1.2.2 Tight Clustering Algorithm

The tight clustering algorithm is as follows:

1. Subsample 70% of the data and assign clusters based on prespecified k_0 clusters. It is

suggested to set k_0 within 1 – 2 times the presumed true value of k. A larger value of k_0 will result in smaller tight clusters.

2. Repeat B times. It is suggested to let $B = 10$. A $\overline{\mathbf{D}}$ matrix tells the proportion of times that two elements belonged to the same cluster.
3. Any set of points with $\overline{\mathbf{D}} \geq 1 - \alpha$ belong to set V. Order the sets $V_{k_{01}}, V_{k_{02}}, \ldots$. Choose the q largest sets (it is suggested to let $q = 7$). α controls the tightness of the clusters.
4. Repeat for consecutive k. Stop when $\frac{|V_{k',i} \cap V_{(k'+1),j}|}{|V_{k',i} \cup V_{(k'+1),j}|} \geq \beta$ where $|V|$ = size of V and β controls the stableness of the selected clusters.
5. The set, $V_{(k'+1),j}$, is a tight and stable cluster. Remove it from the data, reduce k_0 by one and then repeat steps 1 to 4.

Also mentioned above, K–means weights all features equally. A few methods have been developed recently to perform Cluster analysis with feature selection to address this issue. Pan and Shen [2007] use a model-based approach to clustering with an L_1 penalty used to remove non-informative variables from the resulting model. In model-based clustering, a mixture model is assumed where each component of the mixture distribution corresponds to a cluster. The initial MLE for the parameters is determined via the EM algorithm, where the initial parameter values are based on the cluster centers from a K–means algorithm with random initial starting points. In an attempt to avoid the possibility of achieving a local maxima, the EM algorithm is repeated multiple times for any particular combination of numbers of clusters and tuning parameter values. The observations are then assigned to the cluster for which they have the highest probability, based on the MLE estimates.

Pan and Shen included an L_1 penalty with a tuning parameter to the mixture model in order to obtain a sparse solution. Many models are run with different values of the tuning parameter and different numbers of clusters. They use the BIC to determine the best model and the resulting number of clusters. However, this model-based approach is built on the unlikely assumptions in the high dimensional setting that there is the same covariance matrix structure across clusters and independence between variables. Additionally, it assumes that the data follows a Gaussian distribution, which is difficult to validate. The BIC is also an unstable method of model selection with the high likelihood of convergence to only a local maximum [Tseng and Wong, 2005].

1.3 SPARSE K–MEANS

Witten and Tibshirani [2012] developed a sparse K–means algorithm, which accomplishes feature selection by assigning weights to the different features via including the L_1 and L_2 penalties into the K–means clustering algorithm. For $x_{ij} \in \mathbf{X}_{n\times p}$ consisting of n observations and p features, the target function for their algorithm is:

$$\max_{C_1,\ldots,C_k} \left\{ \sum_{j=1}^{p} w_j \left\{ \frac{1}{n} \sum_{i=1}^{n} (x_{ij} - \mu_{.j})^2 - \sum_{k=1}^{K} \frac{1}{n_k} \sum_{i \in C_k} (x_{ij} - \mu_{kj})^2 \right\} \right\} \tag{1.2}$$

where the clusters are denoted as $C_1, \ldots, C_K$, n_k is the number of observations in the k^{th} cluster, μ_{kj} is the k^{th} cluster center for the j^{th} feature for all features $j \in (1, \ldots, p)$ and $w_j \geq 0$, $||w||^2 \leq 1, ||w||_1 \leq s$, where the vector, $\mathbf{w}$, contains the weights for each feature.

The L_2 penalty ensures that there will be more than one non-zero element of w. The L_1 penalty forces the sum of all of the weights to be less than a tuning parameter, s. This results in sparsity when there are small values for the tuning parameter. This tuning parameter is determined by an approach similar to the gap statistic [Tibshirani et al., 2001]. However, the authors acknowledge that this method of selecting the tuning parameter is not very accurate, as it tends to overestimate the number of variables that should remain in the model.

Features with larger between cluster sums of squares are given larger weights. Although the resulting clustering is still associated with only a local minimum, for any given final clustering, the weights assigned to the variables is a convex solution provided an initial tuning parameter value. They extend their idea to hierarchical clustering as well.

Although these methods are helpful in yielding a sparse clustering solution, a global minimum is still not reached. Witten's method, like the standard K–means algorithm is also only applicable after there is a predetermined number of clusters. As noted above, this is particularly problematic in the high dimensional setting, where it is challenging to determine the appropriate number of clusters since many of the innovative methodologies to solve this problem are not successful with high dimensions.

1.3.1 Sparse K–Means Clustering Algorithm

1. Set $w_1 = \ldots = w_p = \frac{1}{\sqrt{p}}$

2. Holding w fixed, apply the standard K–means algorithm

3. Holding $C_1, \ldots, C_k$ fixed, optimize with respect to $\mathbf{w}$, as described in Witten and Tibshirani. For the prescribed $C_1, \ldots, C_k$, this solution of weights is convex.

4. Iterate between Steps 2 and 3 until convergence.

Although Witten and Tibshirani succeeded in demonstrating success in reducing the number of uninformative features selected and thus providing a more accurate clustering of the observations when compared to the standard K–means algorithm in a high-dimensional setting, they left much room for improvement. It can be shown that their algorithm often results in many true signal features being weeded out amongst the noise and only a small proportion of the remaining features being actually true signals. Secondly, in scenarios where there is a weak signal between the different clusters, the sparse K–means algorithm often yields many misclassified observations. Lastly, via simulation, it can be demonstrated that there is often high variability in the number of features selected. This leaves the user lacking confidence if the sparsity achieved is a true reflection of the underlying true features. We look to improve upon these areas via an innovative algorithm, Stable Sparse K–means. We intend to address all of these weaknesses.

Bi et al. [2011], building off of the concepts suggested in Tseng and Wong's (2005) tight clustering algorithm, combined resampling with Witten's feature selection approach to achieve confidence intervals in the K–means clustering solution. However, this methodology has only been applied to a two-cluster setting where the identity of the clusters is already predetermined. It is not clear how to predict the number of clusters or how to identify the final clusters in a multiple cluster solution. It also only provides classification on a small subset of the observations.

1.4 STABILITY ANALYSIS

Tibshirani and Walther [2005] introduced a novel approach to predicting the number of clusters via prediction strength of the clustering. Instead of determining the number of clusters based on final performance of the best solution based on such parameters as within clusters sums of squares, which was the basis for many of the methodologies developed until then [Milligan and Cooper, 1985, Tibshirani et al., 2001], Tibshirani used consistency as his main criteria. Thus, the true number of clusters should reflect the clustering algorithm which consistently groups its observations into the same clusters. He, similar to Tseng, makes use of a co-membership matrix used to summarize the proportions of times in which observations are grouped together. Tibshirani uses two-fold cross validation to determine the prediction strength of the different clustering algorithms. Prediction strength, $ps(k)$ is defined by:

$$ps(k) = \min_{1 \le j \le k} \frac{1}{n_{k_j}(n_{k_j} - 1)} \sum_{i \neq i' \in A_{k_j}} \mathbf{D}[C(X_{tr}, k), X_{te}]_{ii'} \tag{1.3}$$

where $C(X_{tr}, k)$ refers to the clustering algorithm performed on data $\mathbf{X}_{tr}$ with k clusters and $\mathbf{D}[C(X_{tr}, k), X_{te}]_{ii'} = 1$ if after performing the same clustering algorithm on $\mathbf{X}_{te}$, the observations i and i' which previously belonged to the same cluster (when K–means was performed on $\mathbf{X}_{tr}$) would also belong to the same cluster if predicted using the results from $\mathbf{X}_{te}$. Otherwise $\mathbf{D}[C(X_{tr}, k), X_{te}]_{ii'} = 0$. $A_{k_1}, \ldots, A_{k_k}$ are the indices of the test observations in clusters $1, \ldots, k$ and $n_{k_1}, \ldots, n_{k_k}$ are the numbers of observations in these clusters. Thus $ps(k)$ is a reflection of the cluster with the minimum prediction strength from the clustering solution for a given k.

To work around the common problem that a smaller amount of clusters is more inclined to have a higher prediction strength, Tibshirani suggests a criterion for selecting the number of clusters based on the highest number of clusters which yield a prediction strength above a certain cutoff. He suggests using a cutoff between 0.8 and 0.9.

With this methodology, Tibshirani also provides a framework to see how different the clusters are, or whether there is an overlap between the clusters. However, much like the methods suggested prior to Tibshirani's prediction strength method, when the clusters are not well separated his method does not perform well. Also, although in his paper Tibshirani

asserts that his method performs well in a high dimensional setting, his simulations only demonstrate this in a case where 10 % of the features are true signal and they are very well separated. However it can be shown that as the proportion of true signal decreases and the signal distinguishing the clusters gets weaker his methodology does not perform well either.

Steinley [2006] indicated that the number of unique local optima observed over many initializations is an indication of the quality of the clustering solution. Based on this realization, Steinley [2008] proposes a stability analysis to predict the correct number of clusters as well as to best describe and summarize K–means clustering results. He suggests initializing the K–means algorithm many times and then to summarize the results in a co-occurrence matrix where each element in the matrix reflects the proportion of times that each pair of observations belonged to the same matrix.

Assume a K–means algorithm was run R times on a data set where there are N observations and K clusters. Let $A^{(r)}$ be an $N \times N$ matrix with elements $a_{ij}^{(r)} = 1$ if observations i and j belong to the same cluster at the r^{th} partitioning of the observations and let $a_{ij}^{(r)} = 0$ otherwise. Then the consensus matrix, $\mathbf{A}^{(\overline{\mathbf{R}})} = \frac{1}{R}\sum_{r=1}^{R} A^{(r)}$. In order to recognize the strongest clusters, he then suggests to reorganize this matrix to make it block diagonal where the sum of the co-occurrences in each block is maximized. Let Ω refer to the sum of co-occurrences within a given block. Then, $\Omega = \sum_{k=1}^{K} \frac{\sum_{i \in B_k} \sum_{j \in B_k} a_{ij}^{(\overline{\mathbf{R}})}}{n_k} \forall i \neq j$ where B_k is the k^{th} block and n_k is the number of observations in the k^{th} block. To maximize the sums in each block, he uses a quadratic assignment, which is described in detail in Steinley [2008]. Alternatively, Monti et al. [2003] suggests using hierarchical clustering to achieve the block diagonal form.

Once in the block diagonal form, in low dimensions, it is easy to visualize the results of all initializations from the K–means clustering algorithm. Steinley also develops a statistic to determine the correct number of clusters. Suppose $\mathbf{X}$ is the data, then $\Psi_k = \frac{\Omega_k}{\Lambda_k * K}$ where $\Lambda_k = \frac{\sum_{k=1}^{K} \sum_{i \in B_k} \sum_{j \notin B_k} a_{ij}^{(\overline{\mathbf{R}})}}{N}$ which is the between cluster co-occurrence for cluster k and Ω_k is the within cluster co-occurrence (defined above) for cluster k. For each variable, j, in $\mathbf{X}$, generate $\mathbf{X}^* \sim$ UNIF (LB_{x_j}, UB_{x_j}) where LB_{x_j} and UB_{x_j} are the minimum and maximum

values observed for variable j respectively. Then calculate Ψ_k^*. The number of clusters is chosen to maximize $\Psi_k - \Psi_k^*$. If $\Psi_k - \Psi_k^* < 0$ for all k, then there is only one cluster. Based on this method, the correct number of clusters will be chosen based on the solution with the fewest locally optimal solutions. He also adds that if Ψ approaches the maximum possible, $(N^2 - N * K)/K^2$ then one should look more closely at each possibility.

Steinley goes on to describe how to summarize a co-occurrence matrix so that it can be useful even in high dimensions. The sums of co-occurrences within each block, Ψ, can be broken down into $\Omega = \Omega_1 + \Omega_2 + \ldots + \Omega_k$. Thus compactness of a cluster can be described by $\frac{\Omega_k}{n_k - 1}$. Similarly, Λ can be broken down into $\Lambda = \Lambda_1 + \Lambda_2 + \ldots + \Lambda_k$. Thus the overlap between clusters in pairwise fashion can be summarized by $\frac{\Lambda_k}{\Lambda}$. Also based on the co-occurrence matrix, one can predict the probability that the i^{th} observation will belong to the k^{th} cluster, $p_{ik} = \frac{\sum_{j \in B_k} a_{ij}^{(\overline{\mathbf{R}})}}{\sum_{j=1}^{N} a_{ij}^{(\overline{\mathbf{R}})}}$. Similarly, the probability of misclassification, i.e. an object that truly is classified in cluster k being classified in cluster k', $p_{kk'} = \frac{\sum_{i \in k} p_{ik'}}{n_k}$.

Building off of Steinley's stability analysis, instead of looking for stability amongst the clustering of observations, we would like to consider looking for stability amongst the selection of variables resulting from Witten and Tibshirani's sparse K–means method [Witten and Tibshirani, 2012]. We look to take advantage of the tendency for the gap statistic approach to tuning parameter selection to select a tuning parameter which overestimates the number of features which should play a role in the clustering under high dimensions. We suggest repeating the sparse K–means algorithm a large number of times and only consider the features which consistently receive a weight, $w_j > 0$ a large proportion of times. Thus we rely on a conservative initial selection of features from each iteration of the sparse K–means algorithm, by assuming they include a subset of true features irrespective of the noise that is selected amongst them, and via resampling we weed out the noise. By doing so, we hope to demonstrate better accuracy in yielding a subset of features containing true signal, fewer misclassifications amongst observations even in settings where there is a weak

signal distinguishing between clusters, and less variability in the number of features selected amongst datasets from similar distributions.

The concept of applying a stability analysis for feature selection has been introduced by Meinshausen and Bühlmann [2010]. They developed a general method for feature selection applied to a supervised setting in which a model is run and a penalty is added with a tuning parameter attached to induce feature selection. Their agenda was to bypass the need to accurately select a tuning parameter, by instead resampling the data with a range of tuning parameters and only keep the features consistently selected across the entire range of tuning parameters. Their algorithm is as follows:

1.4.1 Algorithm for Stability Selection of Features

1. For each $\lambda \in \Lambda, \hat{s}^{\lambda} \subseteq (1, \ldots, p)$ where there are p features and Λ is the initial set of tuning parameters.
2. Let S(I) be a subsample without replacement of size $\frac{n}{2}$. For every set $K \subseteq (1, \ldots, p)$, the probability of being in the selected subset $\hat{S}^{\lambda}$ is $\hat{\pi}_k^{\lambda} = Pr(K \subseteq \hat{S}^{\lambda}(I))$.
3. The "stable" variables are then selected, $\hat{S}^{stable} = (k : \max_{\lambda \in \Lambda}(\hat{\pi}_k^{\lambda}) \geq \pi_{thr})$ where π_{thr} is a threshold tuning parameter for the proportion of times the stable variable must be selected to be considered stable.

The authors contend that their methodology is computationally more efficient than the standard tuning parameter selection methodology via the bootstrap. This is advocated as well in Valdar et al. [2009]. They also assert that although they are still left with a tuning parameter, π_{thr}, the results do not differ much between sensible choices in a range of the cut-off. In their applications they recommend a value between 0.6 and 0.9. We apply this concept to sparse K–means clustering in an unsupervised setting.

2.0 STABILITY ANALYSIS OF SPARSE K–MEANS

2.1 INTRODUCTION

Amongst the prominent methodologies for feature selection in an unsupervised setting is sparse K–means, developed by Witten and Tibshirani [2012]. Although it provides a dramatic improvement over the standard K–means algorithm in a high dimensional setting, there is still room for improvement. Often many noise features are still selected, there is much variability in the features selected, and there are many misclassified observations when the clusters are very close together. We intend to improve upon this methodology via a stability analysis focused on feature selection. We name this new method Stable Sparse K–means.

The basic concept is that although Witten's method often selects many noisy features, the noise features selected will vary whereas the true signal features will consistently be selected. Thus, if we were to resample our observations and perform sparse K–means a large number of times, the features selected a high proportion of times will likely be the true signal, whereas the rest will be noise. After weeding out the noise, we will then perform standard K–means to classify the observations. This is because we assume all of the remaining features are true signal and it has been demonstrated that in a situation with no noise, standard K–means performs better than sparse K–means [Witten and Tibshirani, 2012].

To start, we intend to apply our new methodology in a setting similar to Witten's sparse K–means paper, where the number of clusters, K is known. We hope to, in the future, adapt our methodology to include scenarios where the number of clusters is unknown. For now, we suggest using one of the above established methods to predict the number of clusters prior

to applying Stable Sparse K–means [Tibshirani et al., 2001, Fang and Wang, 2012, Monti et al., 2003, Steinley and Brusco, 2011, Tibshirani and Walther, 2005, Zou and Hastie, 2005].

First, we will take B subsamples containing 50% of the observations. The reason we choose to take subsamples without replacement, instead of a bootstrap with replacement is that the bootstrap will yield datasets with identical replicate items, which can artificially inflate the compactness of the resulting dataset. Our approach is recommended in Monti et al. [2003]. Although the sample size will be smaller, the number of observations has been shown to have little impact on clustering performance [Monti et al., 2003, Tibshirani and Walther, 2005, Steinley, 2006]. Additionally, in Meinshausen and Bühlmann [2010], they suggest using subsampling instead of bootstrap for their stability selection because of the gain in computational efficiency and the same can be said for our application.

Each subsample contains one half of the observations as described in Meinshausen and Bühlmann [2010]. It is a large enough proportion of the data to yield results reflective of the complete dataset while at the same time it is small enough to enable some variability amongst the subsamples. We chose to take $B = 100$ subsamples because that was the number recommended in Meinshausen and Buhlmann.

We run the sparse K–means algorithm, using the gap statistic as described in Witten and Tibshirani [2012] to select the tuning parameter s on each of the 100 subsamples. Any feature assigned a weight $w > 0$ via the sparse K–means algorithm more than π_{thr} proportion of times will be considered a stable feature.

An exact value for π_{thr} is difficult to set. Based on preliminary experimentation, we have found that relatively small values still achieve sparse solutions, whereas for higher values although they also perform well when clusters are distinguished by a strong signal, however when there is a weak signal distinguishing the clusters, the algorithm tends to weed out too many features. Thus we ran our simulations using π_{thr} values between 0.2 and 0.4. However with the real datasets, which started with considerably more features, we expanded our π_{thr} values to range between 0.2 and 0.9. Our method is flexible in that it can be applied across a range of tuning parameters without a large cost in computational efficiency and the researcher can decide how strict (or sparse) he wants to have for his results.

An additional advantage of using low values for π_{thr} is that it allows us to be conservative in weeding out noise features so that we can have the flexibility to run our algorithm again,

and perhaps weed out the remaining noise. We have found through preliminary experimentation that a second run of our algorithm has the most impact with relatively higher values for π_{thr} and we ran our simulations with π_{thr} values ranging between 0.5 and 0.7 and did so with the real data as well. We have not found any improvement in performing a third iteration and thus only consider at most two iterations.

Our algorithm is described in Section 2.1.1:

2.1.1 Algorithm for Stability Analysis of Sparse K–Means

1. Randomly choose a subsample with 50% of the observations without replacement from the original dataset $X_{n\times p}$, where there are n observations and p features per observation.
2. Run sparse K–means on the subsample, obtaining weights, w_j, for all features $j \in (1, \ldots, p)$
3. Assign to each feature
$$d_{1j} = \begin{cases} 1, & \text{if } w_j > 0 \\ 0, & \text{if } w_j = 0 \end{cases} \tag{2.1}$$
4. Repeat steps 1-3 B times
5. Keep all features, j, with $\frac{1}{B}\sum_{i=1}^{B} d_{ij} \geq \pi_{thr_1}$. Let F_1 denote the number of features remaining.
6. * (optional) Repeat steps 1-4, but now the dimensions of the original observations in step 1 should be $X_{n\times F_1}$.
7. * (optional) Keep all features, j, with $\frac{1}{B}\sum_{i=1}^{B} d_{ij} \geq \pi_{thr_2}$. Let F_2 denote the number of features remaining.
8. Run standard K–means on dataset, $X_{n\times F_2}$.

Although we have provided an algorithm which includes both iterations, one has the option to leave out steps 6 and 7 and perform our algorithm with only one iteration. In order to best choose which tuning parameter combination to select as the true feature selection and clustering of the observations we calculate the prediction strength for each combination, as described in Tibshirani and Walther [2005]. The combination with the best prediction

strength is then selected. In real data application we rely on prediction strength to make this decision as described above.

2.2 SIMULATION RESULTS AND DISCUSSION

We simulated data as described in Witten and Tibshirani [2012]. The simulated dataset $\mathbf{X}_{n\times p}$ had $n = 60$ observations and p variables per observation. These 60 observations made up 3 groups where each group consisted of 20 observations. Denote each group as C_k where k = 1,2, and 3. These groups only differed in the first $q = 50$ variables. The remaining $p - q$ variables are noise. Thus our 3 clusters were made up as follows:

- For $j = 1, \ldots, q, X_{ij} \in C_k$, $X_{ij} \sim N(\mu_k, 1)$, where $\mu_1 = -\mu$, $\mu_2 = 0$, and $\mu_3 = \mu$. Thus each cluster is normally distributed with a mean μ and a standard deviation of 1. The smaller the value of μ the larger the overlap between clusters.
- For $j = q + 1, \ \ldots, \ p$, $X_{ij} \sim N(0, 1)$ for all 3 clusters. Thus these variables should not provide any information towards the true cluster membership of each observation.

In Witten and Tibshirani, they ran simulations for $p = 50$, 200, 500, and 1000 but we only focused on the last two cases of $p = 500$ and $p = 1000$. We picked scenarios similar to Witten, with $\mu = 0.5$, 0.6, 0.7, 0.8, and 0.9, all of which were included in Witten's simulation except that we added the case where $\mu = 0.5$. We made use of Witten's "sparcl" package when running the sparse K–means algorithm in R. We replicated 32 datasets for each combination of p, μ, π_{thr_1}, and π_{thr_2}. There were 60 different combinations. Each dataset was subsampled $B = 100$ times.

For the scenario when $p = 500$, via preliminary experimentation, we found best performance when $\pi_{thr_1} = 0.3$ or 0.4 and thus only simulated for these values. Whereas when p = 1000, we found best performance when $\pi_{thr_1} = 0.2$ or 0.3 and simulated for those values. We included the full range of $\pi_{thr_2} = 0.5$, 0.6, and 0.7 for all scenarios. The reason there was success with higher values for π_{thr_1} in the case where p= 500 than when $p = 1000$ is because there was a greater proportion of true signal when $p = 500$.

Our goal in running the simulations was not only to demonstrate our method's superiority over sparse K–means, but also to perform a sensitivity analysis to see how the results differ depending on tuning parameter combinations and if there are gains in running a second iteration.

We look to compare our methods in terms of classification accuracy as well as feature selection. To judge classification accuracy, we use the Classification Error Rate (CER) as described in Witten and Tibshirani [2012], the proportion of misclassified observations. Its value ranges between 0 and 1, where 0 indicates perfect agreement between predicted cluster membership and true cluster membership and a higher value implies a less accurate prediction. Although, we are aware of the limitations of CER [Monti et al., 2003], such as for the case where the number of clusters is misspecified, for the purposes of this simulation study it was chosen, to be consistent with Witten.

There is no single measure to make a thorough comparison between methods in terms of feature selection. We are interested in a number of qualities. An ideal method should minimize the number of noise features selected, maximize the number of true features selected, and not have much variability between datasets in its results. Instead of combining all three criteria to one objective statistic, we chose to make a comparison in each of the criteria separately, but keeping in mind the overall performance as well. Thus, we looked at p_T, the proportion of $\frac{True\ Signal}{Total\ Features\ Selected}$ as well as T, the number of true signal selected, and s_F, the standard deviation of the Total Features selected.

Tables 2.1 and 2.2 show some of our results with the 32 simulated datasets. These tables contain the average CER for each algorithm for the scenario where $p = 500$ and $p = 1000$ respectively. Similar simulations were run where $\pi_{thr_1} = 0.3$ for the scenario where $p = 500$ and $\pi_{thr_1} = 0.2$ for the scenario where $p = 1000$. All of the simulations are summarized in Figures 2.1 and 2.2 below. The values in bold represent the minimum CER for the given scenario.

It can be seen that within the stable sparse K–means method there is minimal discrepancy in the CER amongst the different tuning parameter values. The same holds true for the results not shown. However, there is a clear distinction in the CER between the stable sparse K–means method and Witten's sparse K–means and standard K–means. The stable

Table 2.1: CER Comparison for $p = 500$

	π_{thr_1}		π_{thr_2}			
μ	0.4	0.5	0.6	0.7	Sparse K–means	K–means
0.5	**0.270**	0.271	0.276	0.290	0.362	0.285
0.6	**0.156**	**0.156**	**0.156**	0.158	0.267	0.245
0.7	**0.044**	**0.044**	**0.044**	**0.044**	0.100	0.191
0.8	**0.011**	**0.011**	**0.011**	**0.011**	0.030	0.098
0.9	**0.005**	**0.005**	**0.005**	**0.005**	0.010	0.045

Table 2.2: CER Comparison for $p = 1000$

	π_{thr_1}		π_{thr_2}			
μ	0.3	0.5	0.6	0.7	Sparse K–means	K–means
0.5	0.303	**0.299**	0.303	0.323	0.357	0.318
0.6	0.184	0.184	0.185	**0.182**	0.296	0.271
0.7	**0.068**	**0.068**	**0.068**	**0.068**	0.131	0.237
0.8	**0.011**	**0.011**	**0.011**	**0.011**	0.029	0.186
0.9	**0.001**	**0.001**	**0.001**	**0.001**	0.006	0.110

sparse K–means method has a lower CER for the entire range of μ that was considered. The complete simulation results are presented in Figure 2.1 and Figure 2.2 below.

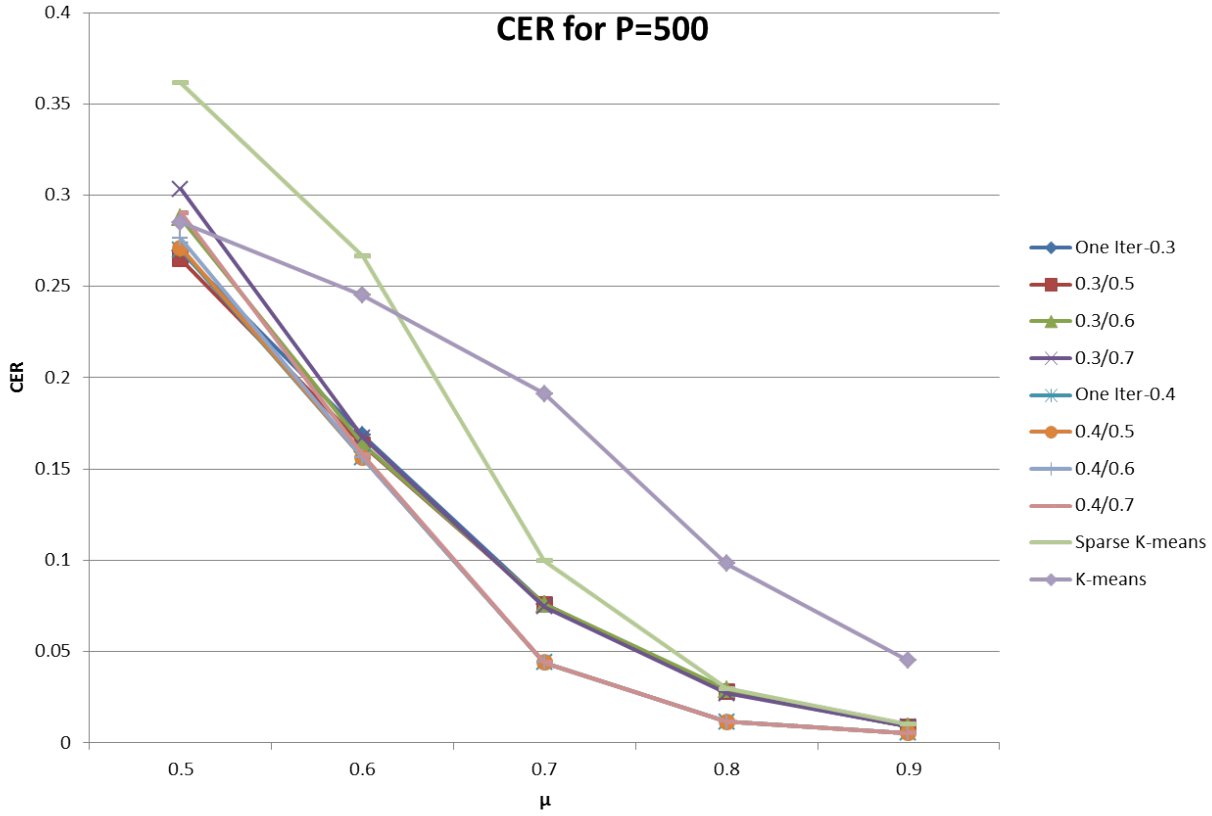

Figure 2.1: The average CER calculated across 32 simulated datasets when p = 500. There is a line representing Wittin's sparse K–means as well as one representing standard K–means. The rest reflect different tuning parameters within the Stable Sparse K–means method. They are labeled as πthr1/πthr2 , except for the lines representing where only one iteration was performed. They contain the values for πthr1 .

As mentioned above we compare the quality of the feature selection between stable sparse K–means and sparse K–means in different dimensions. In Tables 2.3 and 2.4, we present some of our results comparing the proportion of true signal that was selected amongst the stable features and the total number of true signal that was selected amongst the stable features for the scenario when $p = 500$ and $p = 1000$ respectively. It is important to consider

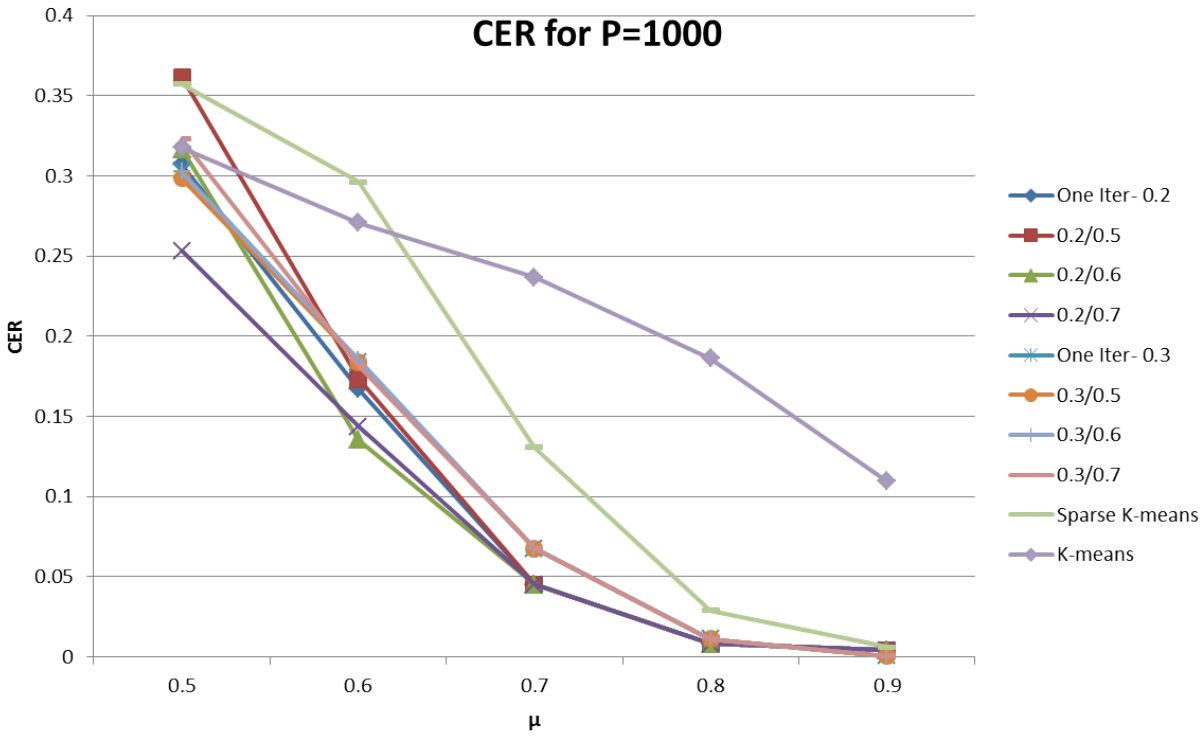

Figure 2.2: The average CER calculated across 32 simulated datasets when p = 1000.

both of these criteria simultaneously as well as independently, as a researcher may be more conservative and prefer to contain as many true features as possible even at the expense of added noise, whereas another researcher may prefer a method which is more accurate.

Table 2.3: Proportion of True Signal Selected, p_T, and the number of true signal features that were selected, T, when $P = 500$. The correct number of true signal in each dataset was 50. The number of True Signal Selected is given in parentheses, p_T (T).

	π_{thr_1}		π_{thr_2}		
μ	0.4	0.5	0.6	0.7	Sparse K-Means
0.5	0.669 (16.7)	0.754 (16.4)	0.805 (15.5)	0.83 (13.6)	0.158 (19.6)
0.6	0.762 (28.6)	0.762 (28.6)	0.763 (28.6)	0.782 (28.6)	0.193 (37.6)
0.7	0.788 (45.3)	0.788 (45.3)	0.788 (45.3)	0.788 (45.3)	0.235 (49.8)
0.8	0.745 (49.6)	0.745 (49.6)	0.745 (49.6)	0.745 (49.6)	0.253 (50)
0.9	0.632 (50)	0.632 (50)	0.632 (50)	0.632 (50)	0.241 (50)

Table 2.4: Proportion of True Signal Selected, p_T, and the number of true signal features that were selected, T, when $P = 1000$. The correct number of true signal in each dataset was 50. The number of True Signal Selected is given in parentheses, p_T (T).

	π_{thr_1}		π_{thr_2}		
μ	0.3	0.5	0.6	0.7	Sparse K-Means
0.5	0.099 (27.2)	0.403 (14.7)	0.63 (10.4)	0.785 (6.3)	0.083 (20.5)
0.6	0.727 (21.4)	0.728 (21.4)	0.735 (21.4)	0.789 (21.3)	0.232 (27.9)
0.7	0.987 (30.9)	0.987 (30.9)	0.987 (30.9)	0.987 (30.9)	0.318 (45.9)
0.8	0.988 (44.6)	0.988 (44.6)	0.988 (44.6)	0.988 (44.6)	0.379 (49.9)
0.9	0.942 (49.5)	0.942 (49.5)	0.942 (49.5)	0.942 (49.5)	0.505 (50)

We found there was hardly any difference between running two iterations and one iteration of our method for most scenarios except for the scenario when there was a very weak

signal amongst the clusters ($\mu = 0.5$). However, we did find, as expected, that our method proved to be significantly more accurate in weeding out the noise than the current sparse K–means method, even in the cases where the clusters were well separated. Although in most cases the current sparse K–means method had slightly higher amounts of true signal selected than in the stable sparse K–means method, the small discrepancy does not make up for the sizable gains in accuracy.

This distinction is brought out by making comparisons within the plots in Figure 2.3 and Figure 2.4. These figures contain the data for all scenarios that were simulated. While one can clearly see the line representing sparse K–means is isolated from the other lines representing Stable Sparse K–means in the plot comparing the proportions, in the corresponding plot showing the number of true signal selected, the discrepancy is much less clear.

In addition to comparing the accuracy of the two methods, we also wish to compare the variability in the number of features selected between the different datasets. For the cases where there is a very weak signal ($\mu = 0.5$), both methods had large variability. However, in almost every scenario, the stable sparse K–means had less variability than sparse K–means. Moreover, when the clusters are a little more separate ($0.7 \leq \mu \leq 0.9$) we found there was considerably less variability in the stable sparse K–means method over the sparse K–means method. Some of these results are presented in Table 2.5 and Table 2.6 below. The complete simulation results are summarized in Figures 2.5 and 2.6 below.

2.3 ANALYSIS OF TWO LEUKEMIA DATASETS

We applied our algorithm to two leukemia datasets: one from Balgobind [Balgobind et al., 2011] and one from Verhaak [Verhaak et al., 2009]. A description of the data is presented in Table 2.7. We only considered samples from acute myeloid leukemia (AML) with subtype inv(16) (inversions in chromosome 16), t(15;17) (translocations between chromosome 15 and 17) and t(8;21) (translocations between chromosome 8 and 21). These three gene-translocation AML subtypes have been well studied with different survival, treatment response and prognosis outcomes. We treat these class labels as the underlying truth to

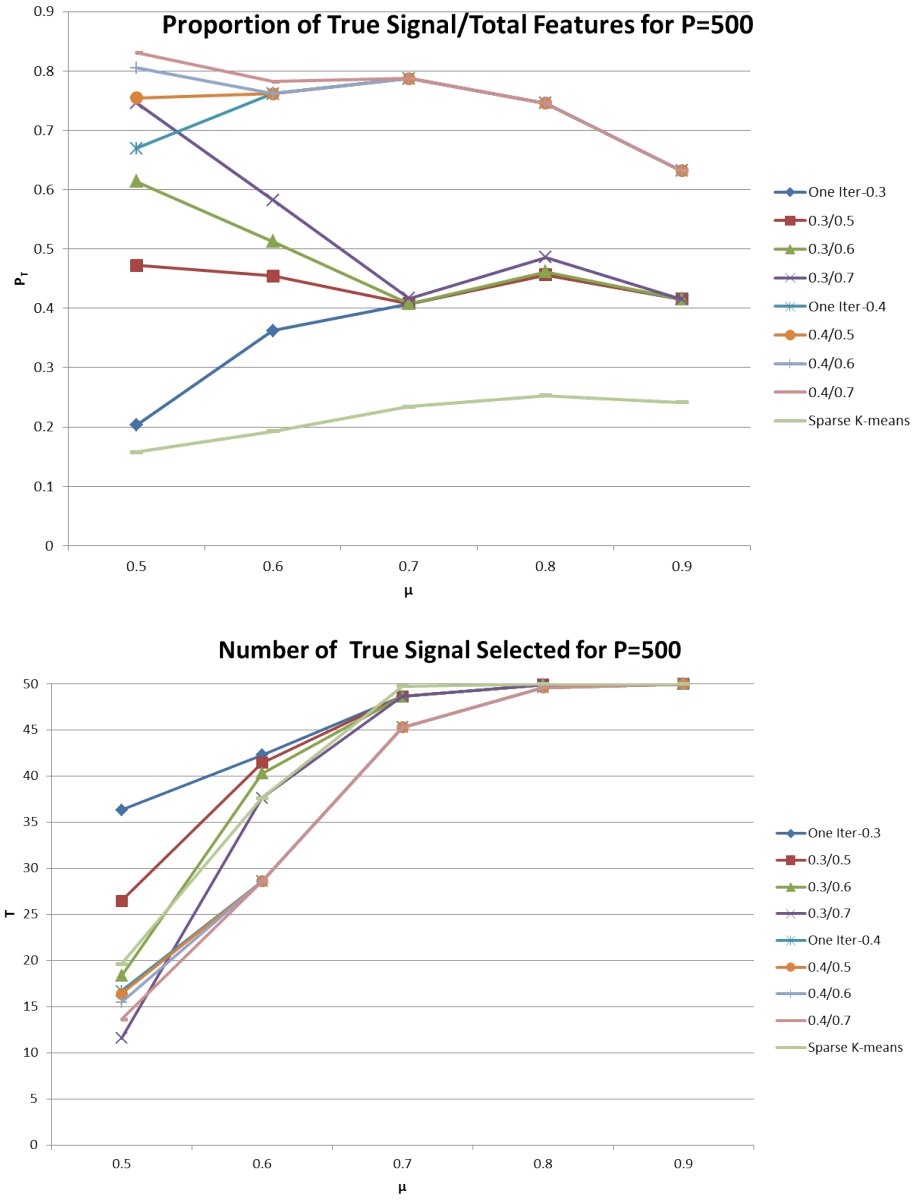

Figure 2.3: The average p_T and T calculated across 32 simulated datasets when $p = 500$. The correct number of true signal in each dataset was 50.

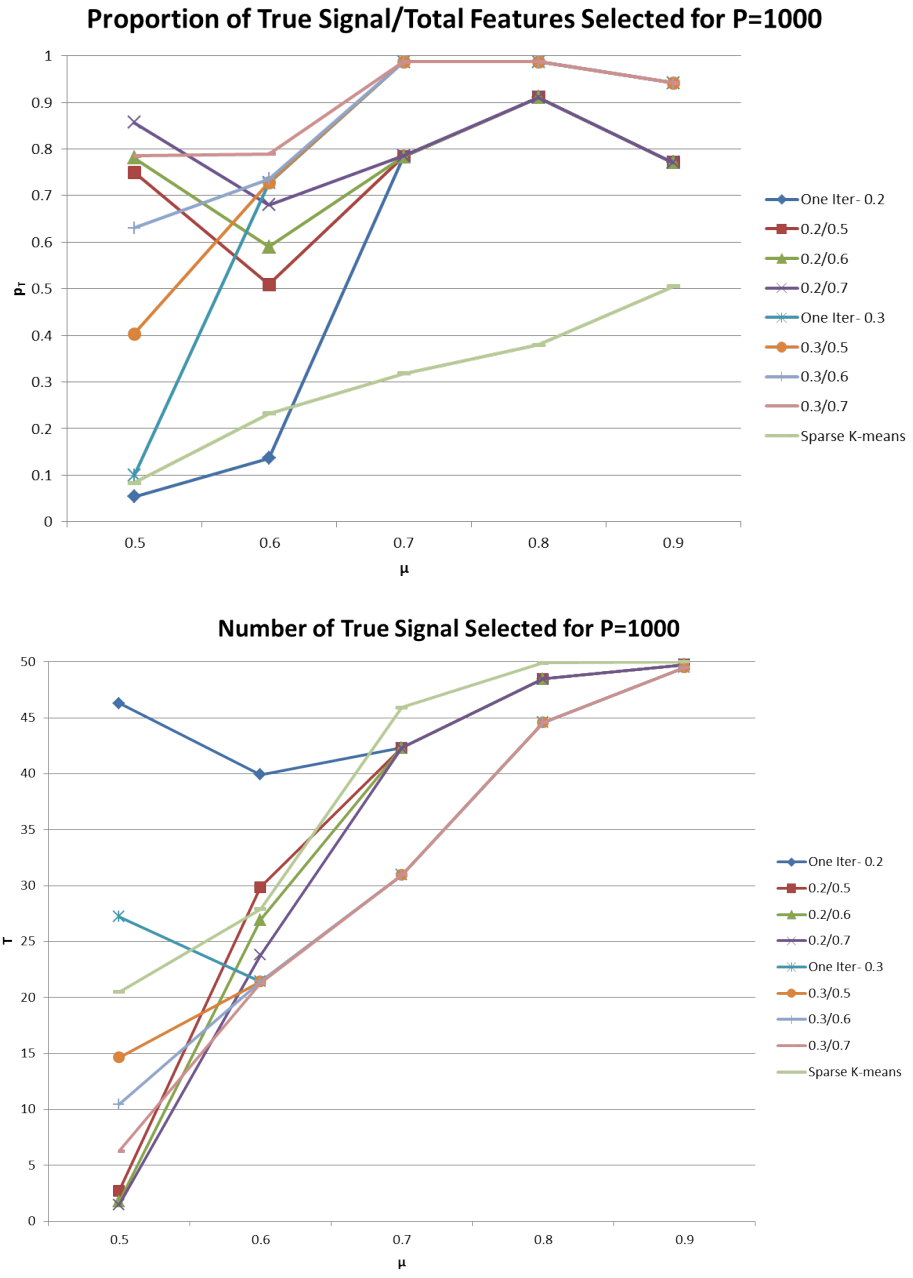

Figure 2.4: The average p_T and T calculated across 32 simulated datasets when $p = 1000$. The correct number of true signal in each dataset was 50.

Table 2.5: The average number of features selected and the standard deviations across 32 simulated datasets for the scenario when $p = 500$. The standard deviation, s_F, of the number of features is given in parentheses.

	π_{thr_1}	π_{thr_2}			
μ	0.4	0.5	0.6	0.7	Sparse K-Means
0.5	24.9 (32.8)	21.8 (20.6)	19.2 (17.8)	16.4 (17)	124.4 (182.9)
0.6	37.5 (35.1)	37.5 (35.1)	37.5 (35)	36.6 (30.9)	195.1 (153)
0.7	57.5 (18.6)	57.5 (18.6)	57.5 (18.6)	57.5 (18.6)	212.3 (92.7)
0.8	66.5 (16.6)	66.5 (16.6)	66.5 (16.6)	66.5 (16.6)	197.9 (57.6)
0.9	79.1 (17.7)	79.1 (17.7)	79.1 (17.7)	79.1 (17.7)	207.2 (18)

Table 2.6: The average number of features selected and the standard deviations across 32 simulated datasets for the scenario when $p = 1000$. The standard deviation, s_F, of the number of features is given in parentheses.

	π_{thr_1}	π_{thr_2}			
μ	0.3	0.5	0.6	0.7	Sparse K-Means
0.5	273.7 (360.4)	36.3 (63.8)	16.6 (23.1)	8 (12.2)	247 (360.9)
0.6	29.5 (31)	29.4 (30.9)	29.2 (30.1)	27 (23.8)	120.2 (204.7)
0.7	31.3 (7.6)	31.3 (7.6)	31.3 (7.6)	31.3 (7.6)	144.2 (181.8)
0.8	45.2 (3.2)	45.2 (3.2)	45.2 (3.2)	45.2 (3.2)	131.6 (136.1)
0.9	52.6 (2.7)	52.6 (2.7)	52.6 (2.7)	52.6 (2.7)	99.1 (12.7)

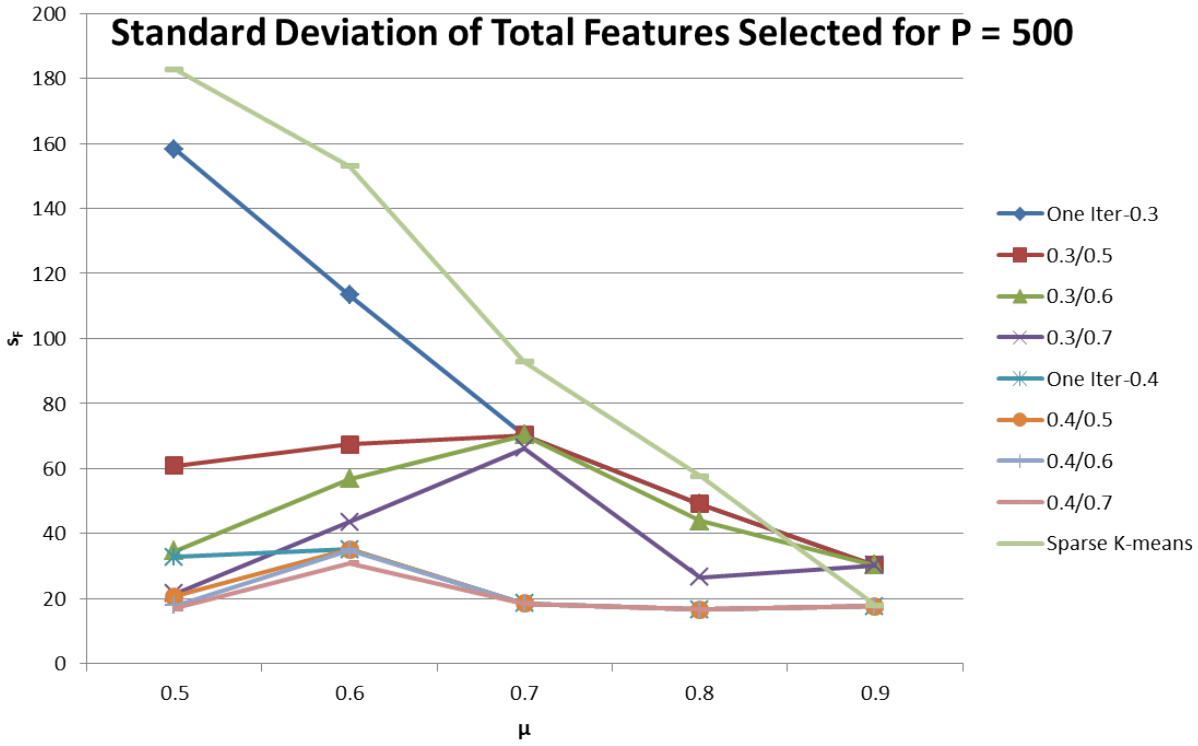

Figure 2.5: The standard deviation, s_F for Total Features selected calculated across 32 simulated datasets when $p = 500$.

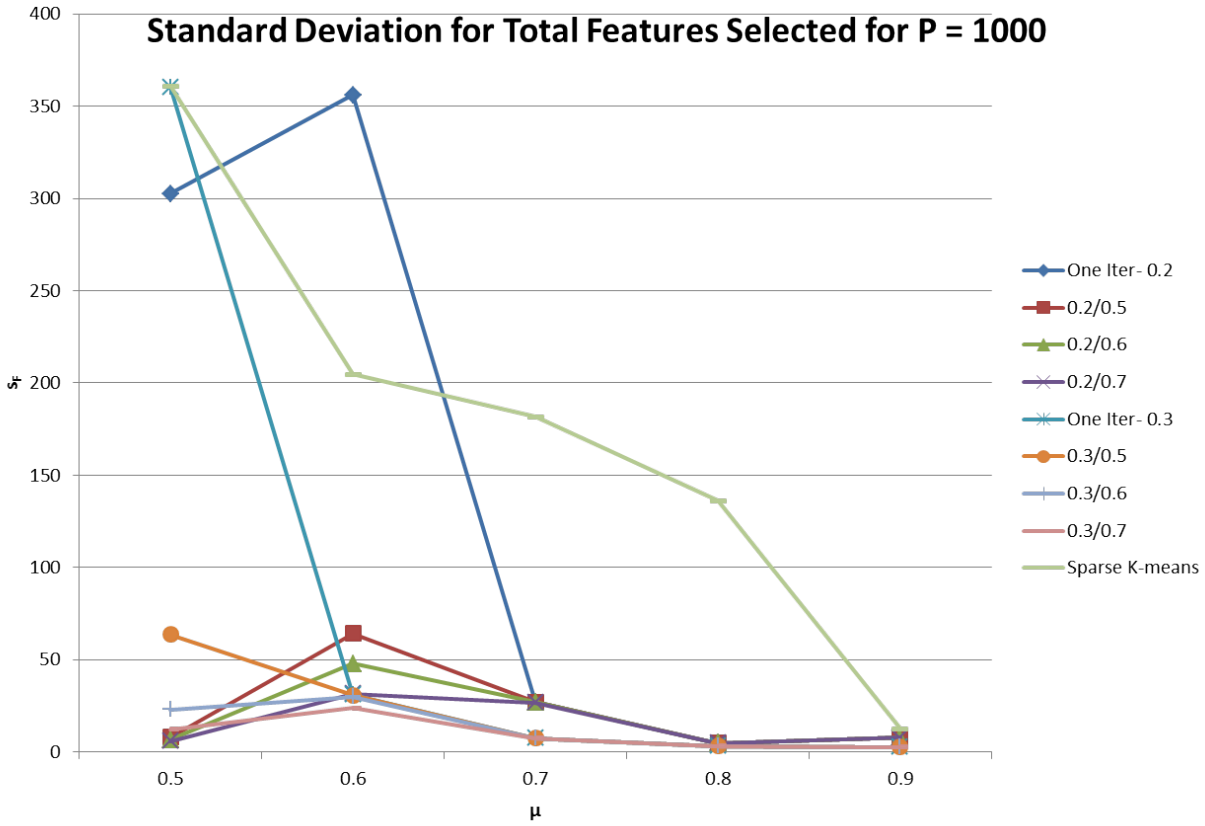

Figure 2.6: The standard deviation, s_F for Total Features selected calculated across 32 simulated datasets when $p = 1000$.

evaluate the clustering performance. The expression data ranged from 3.169 to 15.132. The datasets were downloaded directly from the NCBI GEO website. Originally there were 54,613 probe sets and we filtered out probes with 0 standard deviation in either study. In the end, 48,788 probes remained matched across the studies. Two gene expression matrices with sample sizes 74 and 89 were used as input data for disease subtype discovery.

Table 2.7: Leukemia dataset information

Study Name	Balgobind et al	Verhaak et al
Number of probes	48,788	48,788
Number of patients	74	89
True class label *	(27, 19, 28)	(33, 21, 35)
Data range	[3.169, 15.132]	[4.907, 14.159]
Mean intensity	6.093	6.163
Standard deviation	1.334	1.543
*: true class labels are the number of samples for (inv(16), t(15;17), t(8;21))		

To compare Stable Sparse K–means with sparse K–means, when implementing the sparse K–means algorithm, we selected the tuning parameter s via the minimum gap statistic within one standard deviation of the maximum gap statistic as described in Witten and Tibshirani [2012]. If one would use just the maximum gap statistic with such high dimensions, it assigns small weights to almost all features and no sparsity is achieved. Thus, it is standard practice to use the minimum gap statistic within one standard deviation of the maximum gap statistic in order to achieve sparsity.

We performed each algorithm 35 times and report the average results. We conducted a sensitivity analysis by applying our method across a wide range of values for π_{thr_1} (ranging from 0.2 to 0.9) and are able to show that the results are similar across all of the tuning parameters used. The results from the sensitivity analysis are included below in Figure 2.7 and Figure 2.8.

We also calculated the classification error rates of each method for the two datasets. The sparse K–means algorithm struggled to accurately classify the observations in the Balgobind dataset (CER = 0.29) whereas it was successful in the Verhaak dataset (CER = 0.03). Our method showed it can succeed in both datasets (CER for Balgobind ranged from 0.05 to 0.1 and CER for Verhaak ranged from 0.04 to 0.06). The results are summarized in Figure 2.9.

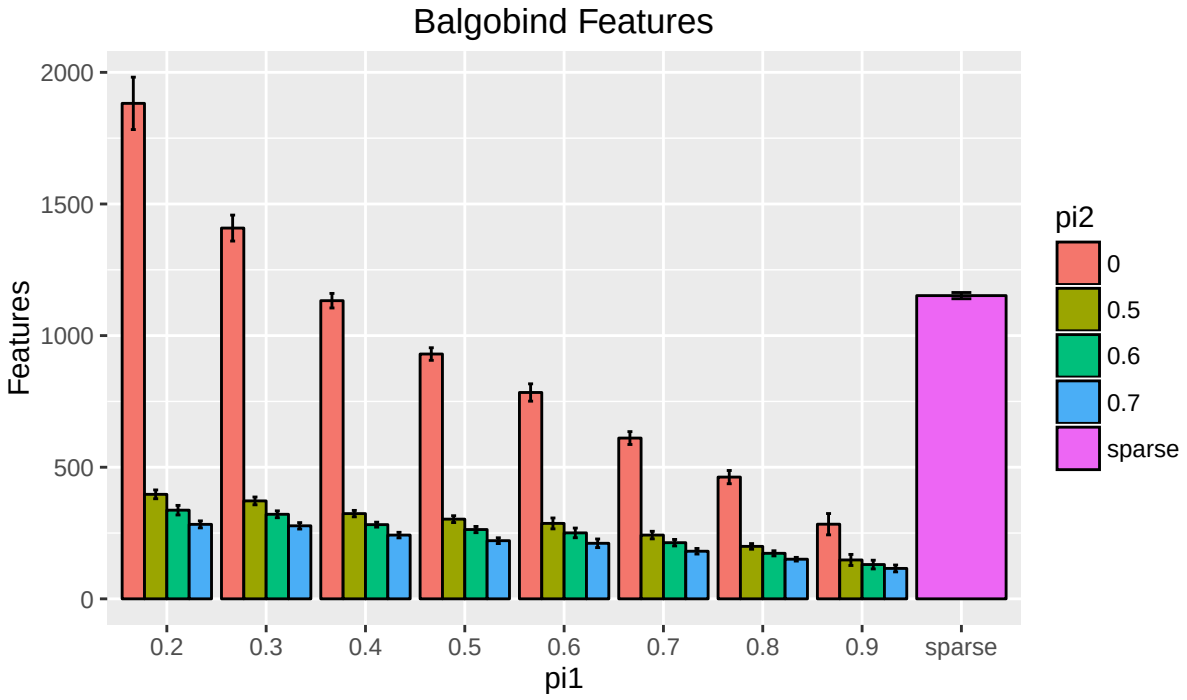

Figure 2.7: The average number of features selected across 35 runs of each algorithm on the Balgobind and Verhaak datasets. **Note**: the bar indicating that pi2 = 0 represents the data from when only one iteration of Stable Sparse K–means was run. There is an error bar at the top of each bar indicating the standard deviation.

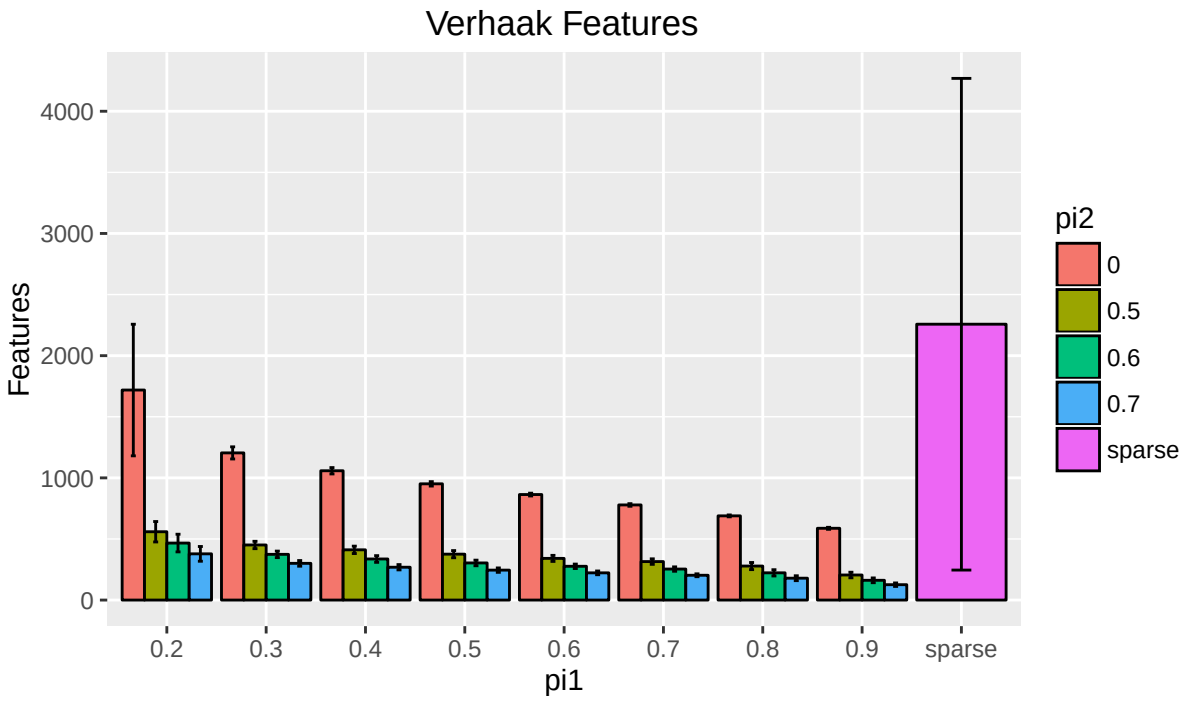

Figure 2.8: The average number of features selected across 35 runs of each algorithm on the Verhaak dataset.

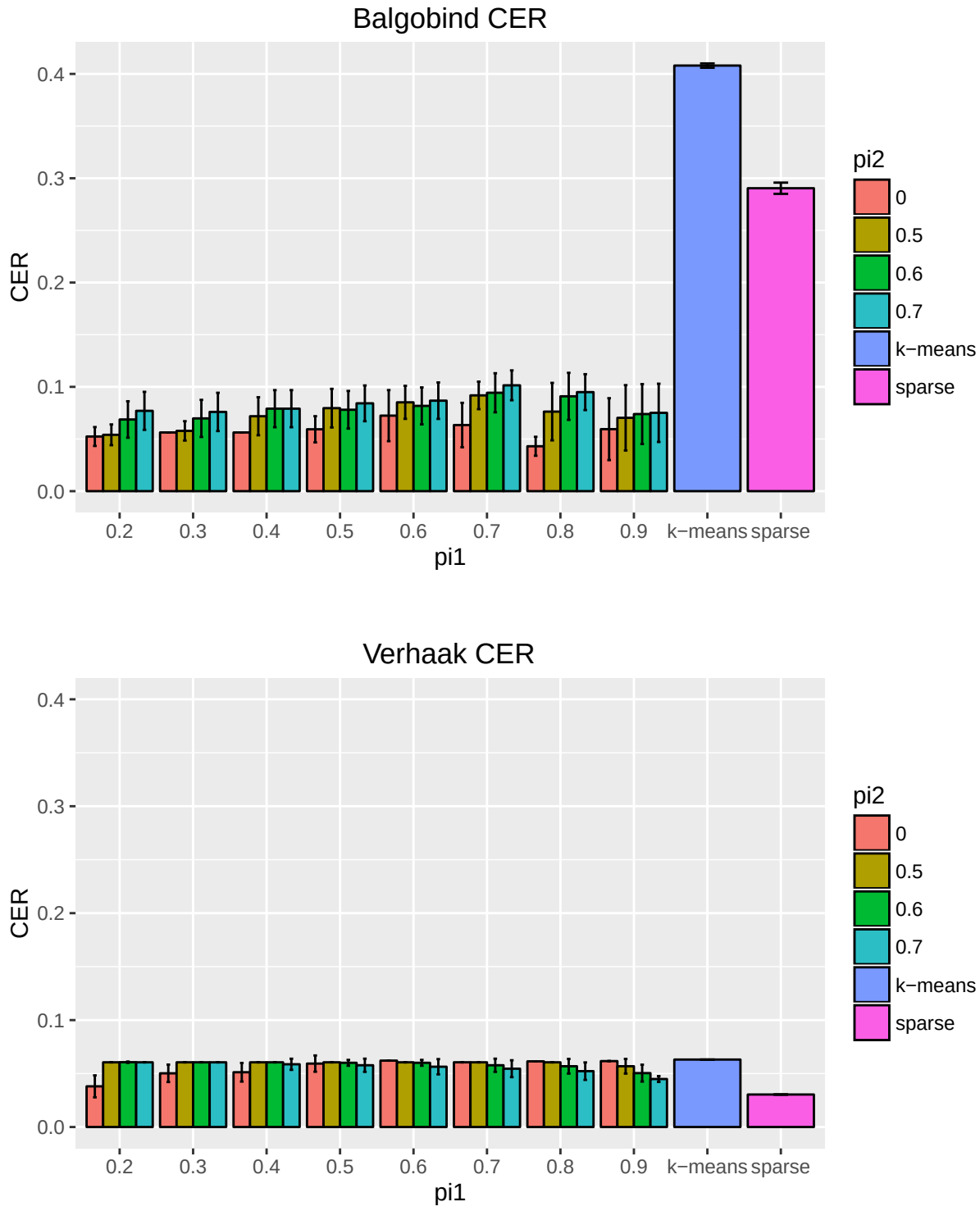

Figure 2.9: The average classification error rate across 35 runs of each algorithm on the Balgobind and Verhaak datasets.

Note: the bar indicating that pi2 = 0 represents the data from when only one iteration of Stable Sparse K–means was run. There is an error bar at the top of each bar indicating the standard deviation.

In addition to validating our method against Witten's sparse K–means using the real leukemia datasets, we also look to compare methods in datasets where there is an increased level of noise. Thus, we added varying levels of white noise to both leukemia datasets by simulating an $N \times p$ matrix from a Normal distribution with a mean of 0 and variance σ^2 and adding this to each existing dataset. However, due to the increased noise level, we only ran our method with tuning parameter π_{thr_1} ranging from 0.2 to 0.5. As the noise level increases less variables will be consistently selected resulting in too much sparsity for high π_{thr_1} values. We used the same π_{thr_2} throughout.

We then compared the CER for both methods on each dataset across values for σ^2 ranging from 0.5 to 2.5. This was performed across 35 simulated noise datasets (stemming from each of the leukemia datasets). The complete results are summarized in Figures 2.10 - 2.13 below. However, for noise level 2.0 for strict tuning parameters the results were too sparse and thus results were only included for small values of π_{thr_1}. Similarly, for noise level 2.5, we only included the results for tuning parameter with highest prediction strength.

In our methodology we suggest to use prediction strength to help determine the correct tuning parameter combination for π_{thr}. We display only the tuning parameter combination which achieved the highest prediction strength. The results are summarized in Figure 2.14 below.

In addition, we make the claim that our method is more stable in its feature selection than sparse K–means. In this context, we define stability as the ability for a method to consistently select a similar subset of features even in scenarios where there is noise added. Viewing the subset of genes selected by each method in the clean dataset as the gold standard, we compare each methods ability of selecting the same subset of genes amidst noise.

To assess stability we applied each method once to each of the clean leukemia datasets. We considered the subset of features selected by each method (for Stable Sparse K–means we selected the tuning parameter combination with greatest prediction strength) as the underlying truth. We then compared the average proportion of genes selected by each method (for Stable Sparse K–means we used the same tuning parameter combination which was used to select the initial subset in the clean dataset) which belonged to the original

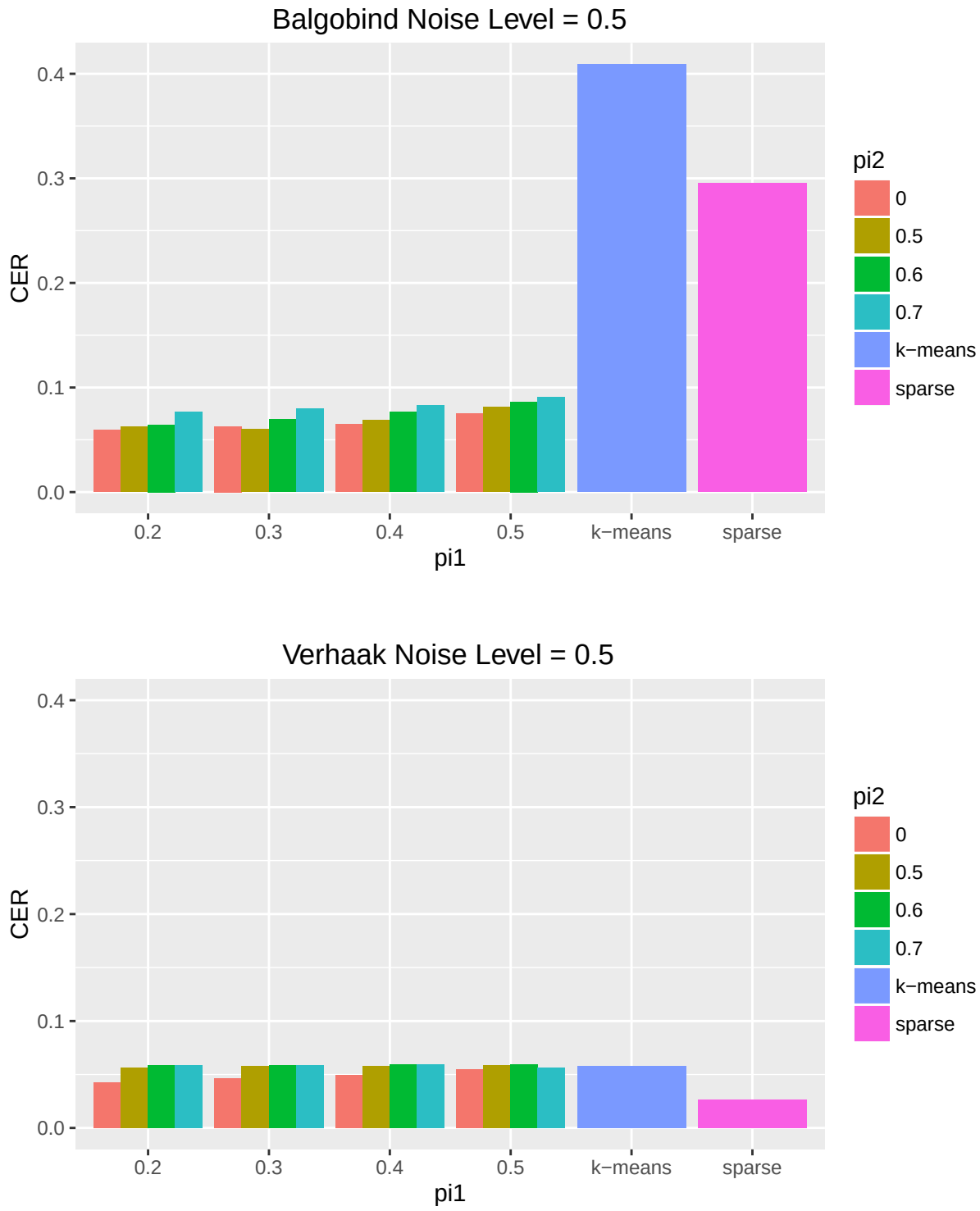

Figure 2.10: The average CER calculated across 35 simulated Balgobind and Verhaak datasets when the Noise Level = 0.5.

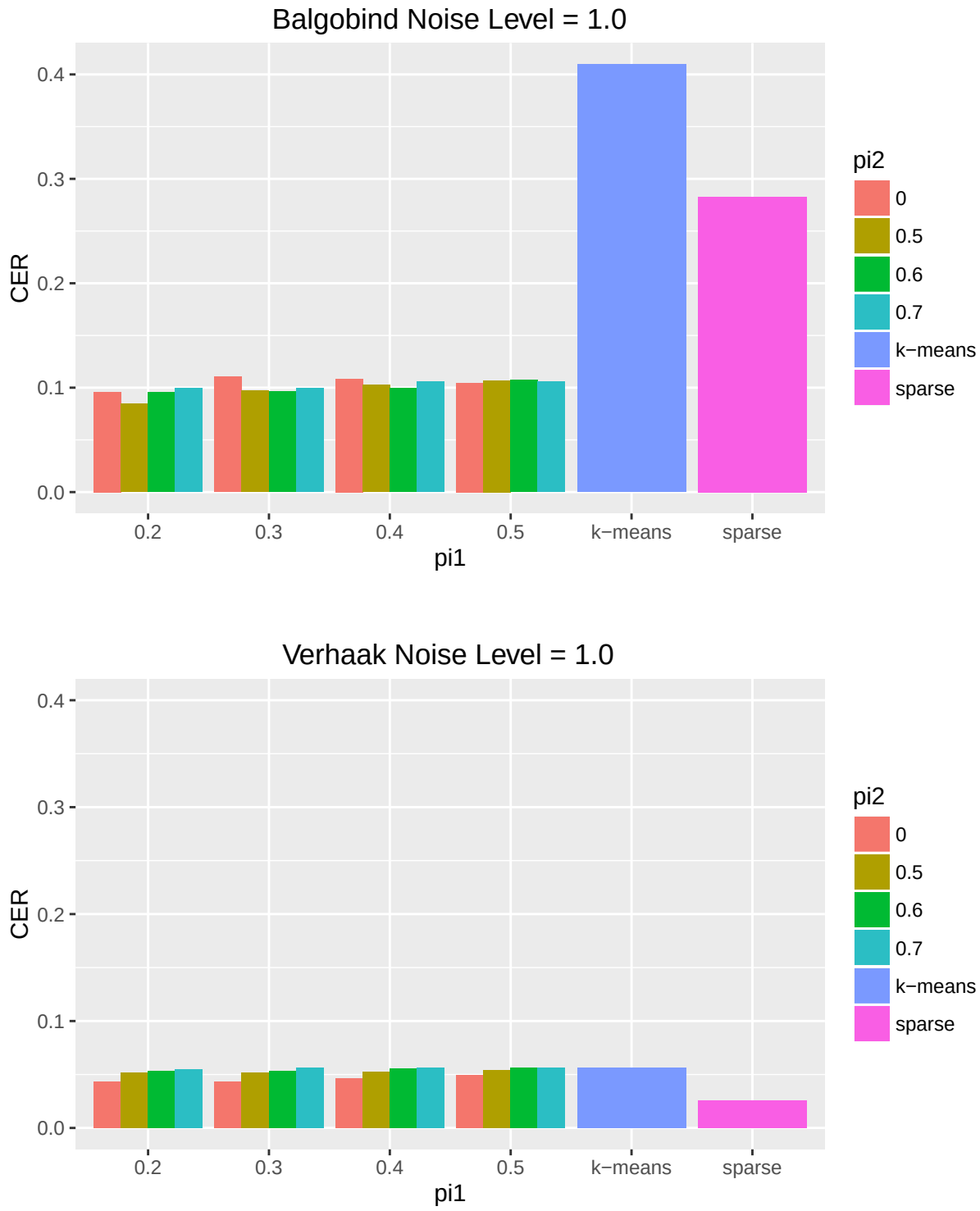

Figure 2.11: The average CER calculated across 35 simulated Balgobind and Verhaak datasets when the Noise Level = 1.0.

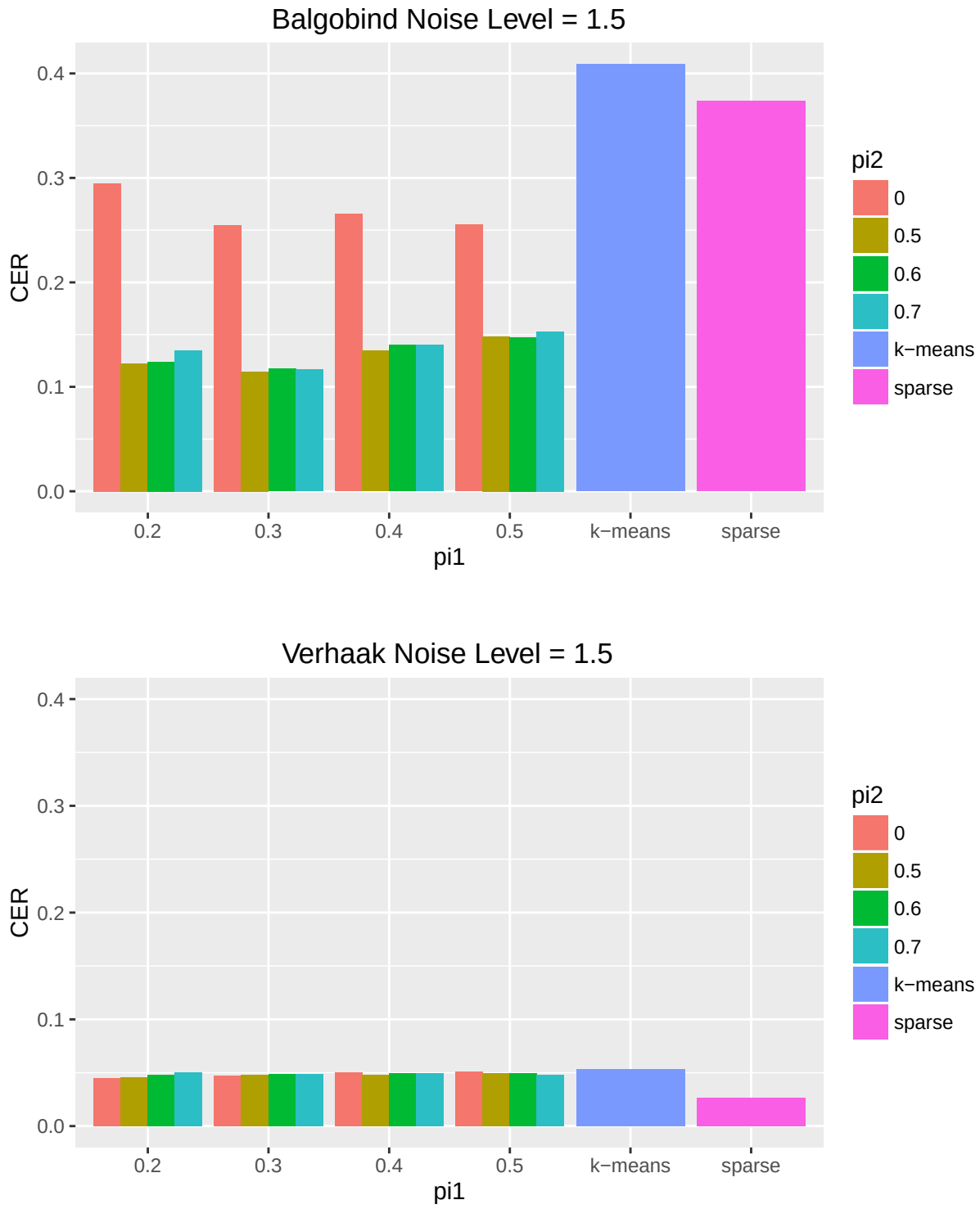

Figure 2.12: The average CER calculated across 35 simulated Balgobind and Verhaak datasets when the Noise Level = 1.5.

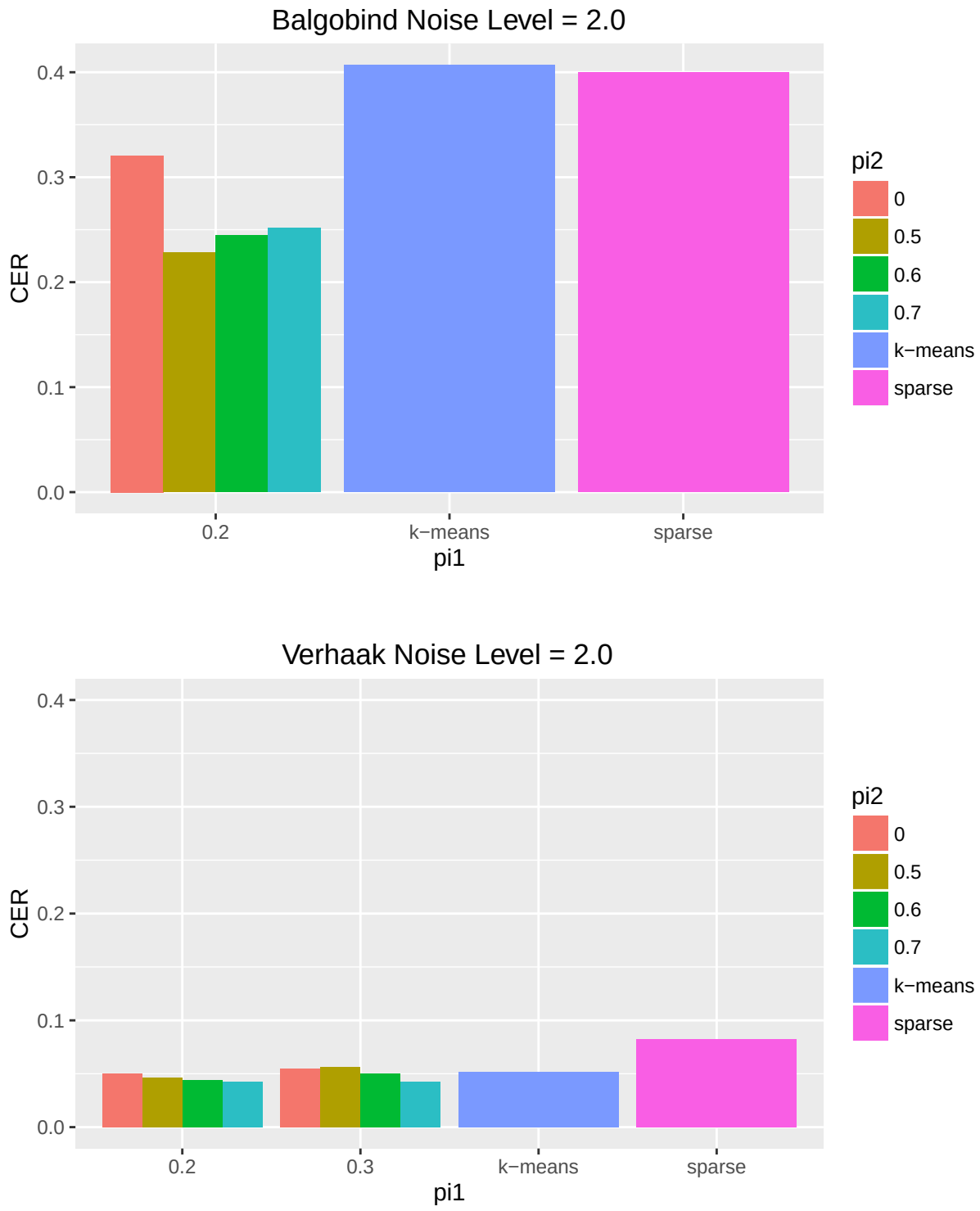

Figure 2.13: The average CER calculated across 35 simulated Balgobind and Verhaak datasets when the Noise Level = 2.0.

Note that the results were too sparse for strict π_{thr_1} values and thus only results for lower π_{thr_1} values can be shown.

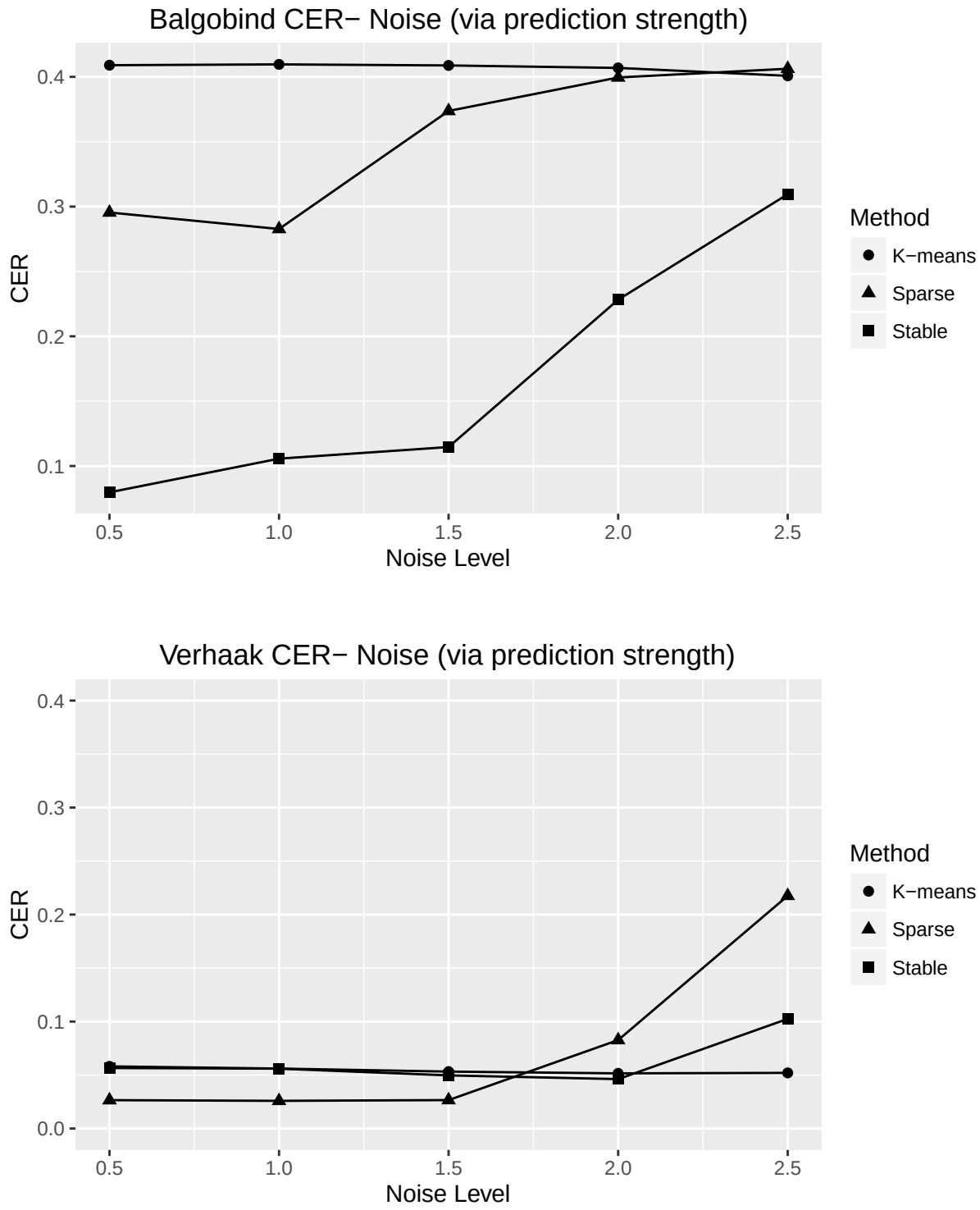

Figure 2.14: The average classification error rate across 35 runs of each algorithm on 35 simulated Balgobind and Verhaak datasets across a range of noise levels.

subset of genes selected in the clean datasets across 35 simulated noise datasets. The results are summarized in Figure 2.15 below.

It has been shown via simulations that Stable Sparse K–means has more accurately classified observations, has accomplished a more accurate selection of a subset of the features, and provided more consistent, less variable results than sparse K–means.

In the real data setting, we have shown that even in a scenario where the sparse K–means algorithm has shown success in classification accuracy, such as in the Verhaak dataset, Stable Sparse K–means has reported similar results. However, in a dataset such as Balgobind, where sparse K–means has faltered, Stable Sparse K–means retained its ability to classify the observations correctly. The differentiation between the two datasets can possibly be explained that in the Balgobind dataset, the underlying centers for the different clusters are close together, producing a weak signal. This would then justify our results as consistent with the simulations where sparse K–means was shown to be inaccurate in cases of weak signal whereas Stable Sparse K–means was more successful.

We demonstrated this claim by adding varying levels of white noise to the datasets. For low levels of noise, the results were similar to the results from the clean datasets. As the noise level increased, however, the sparse K–means method yielded a significantly higher error rate whereas our method remained consistently low.

Additionally, we reinforced our claim to be able to consistently select the same features despite surrounding noise that may be added, whereas the sparse K–means method tended to select subsets of features which differed greatly depending on the noise level. Thus, a researcher can be more confident in the feature selection from our method as reflecting a true subset of features since they would not change regardless of the noise level of the data whereas with the sparse K–means method, the researcher must be wary that if there would be more or less noise in the dataset an entirely different subset of features may have been selected, thus not allowing him to be confident with his results. We have thus succeeded in demonstrating the consistent feature selection of Stable Sparse K–means whereas sparse K–means has been shown to at times offer much variability. Our results on the real leukemia datasets has thus proved consistent with our simulation results.

Our sensitivity analysis from both the simulations and leukemia datasets revealed that there does not appear to be a major distinction between tuning parameter values assuming

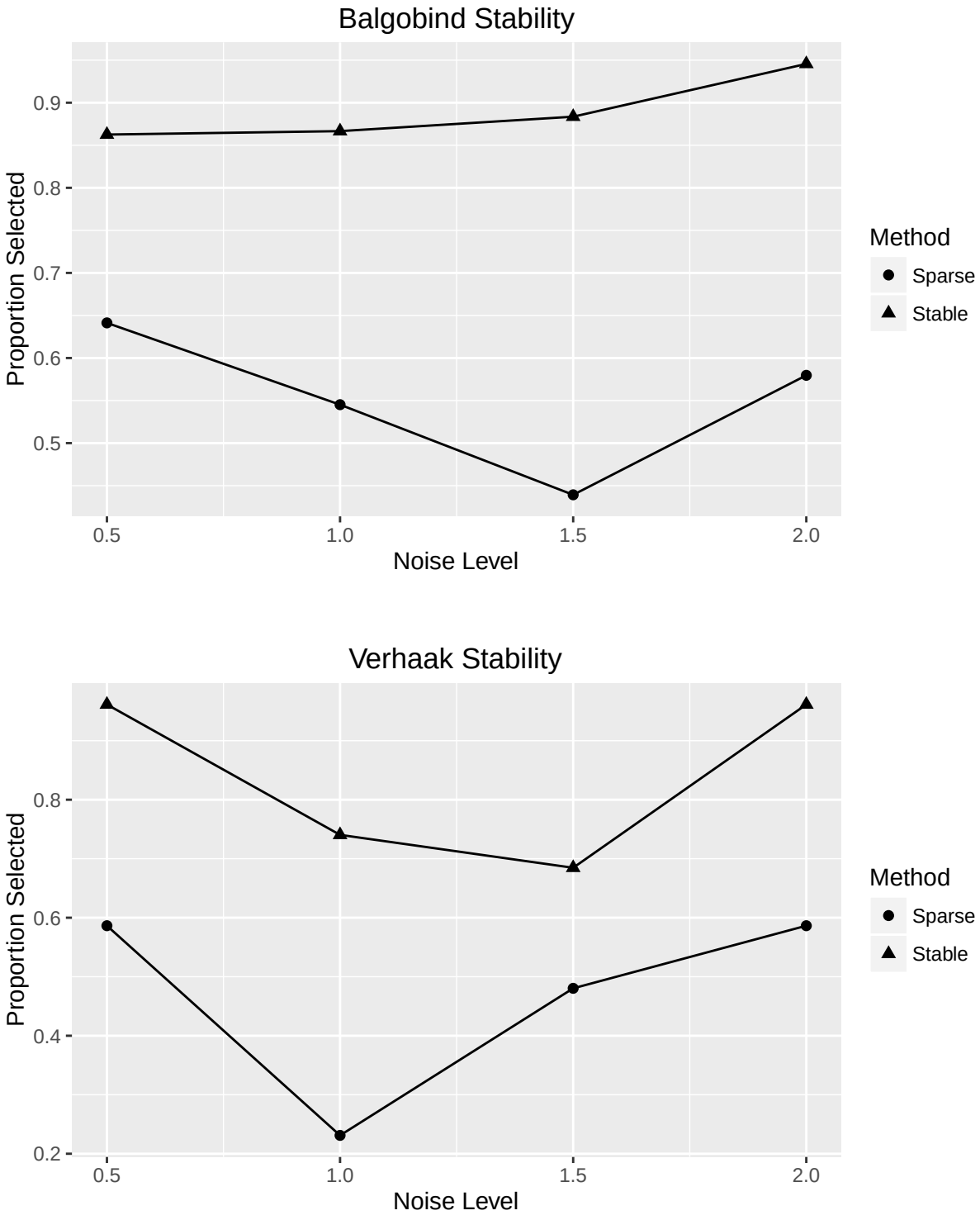

Figure 2.15: The average proportion of genes selected which belonged to original subset of selected features in clean Balgobind and Verhaak datasets across 35 runs of each algorithm on 35 simulated noise Balgobind and Verhaak datasets.

that there are two iterations. However, there did, at times, appear to be distinctions in the results between running one iteration or two iterations. Additionally, in situations of high noise level, even when two iterations were run, our method tended to fail with strict tuning parameter values. Therefore, we suggest to use prediction strength to determine how many iterations to use and decide on the particular tuning parameters.

3.0 FUTURE DIRECTIONS

Wang [2010] develops a method for measuring clustering instability. This method makes use of cross validation to calculate its instability measure. Fang and Wang [2012] develop this method a bit further by calculating the instability measure using bootstrap samples of the data. They show that with the bootstrap their results are more efficient, as they do not have to cut their sample size in half in order to perform a cross validation. However, both of these methods fail when there are redundant features. The authors acknowledge a need to first accomplish feature selection before applying their methodology. They also make mention that their bootstrap method can be used to determine a more accurate value for the tuning parameter in Witten's sparse K–means algorithm. We hope to develop this idea.

Wang suggests that the algorithm which minimizes the measure of instability should reflect the correct number of clusters. To calculate the instability measure, Wang first defines the distance between two clustering algorithms as the probability that two observations will belong to the same cluster in one of the algorithms, but belong to a different cluster in the other algorithm. The instability measure is then the expected value of this distance measure. Thus, a clustering algorithm is considered stable if it is expected that for any given distribution of data, clusterings from different samples will produce similar results.

Wang [2010] projects that if an incorrect number of clusters are specified, the instability statistic will be large. This is because if the specified number of clusters, k, is larger than the true number of clusters, k_t, then the true clusters will be split into small ones and this splitting will vary from sample to sample. If $k < k_t$ then the true clusters will be merged into big ones and this will also change with sampling.

It should be noted, however, that this assertion is not always true. Steinley [2008] provides an example where this would not occur, where there are three well separated clusters

but two of them are close together (although still well separated). If one would perform K–means clustering assuming two clusters, the results will still be very stable even though the true number of clusters is three. Tibshirani and Walther [2005] asserts that a lower assumed number of clusters will usually have a higher prediction strength and therefore suggested choosing the highest k above a certain threshold. We suggest to have a similar criteria in Fang and Wang's method.

Sun et al. [2012] further developed this idea with a methodology known as regularized K–means clustering, by adding an adaptive grouped Lasso penalty onto the cluster centers to remove redundant features. To select a number of clusters and simultaneously perform tuning parameter selection, they suggest using clustering stability. They define instability as above [Wang, 2010] and use the bootstrap method to predict the level of instability, $S(\psi, k, \lambda, n)$, for any particular clustering algorithm with its respective tuning parameters given below.

3.0.1 Algorithm for Selection of Number of Clusters and Tuning Parameters

1. Given n observations, $(X_1, X_2, \ldots, X_n)$, generate three bootstrap samples of size n: z_1^{*b}, z_2^{*b}, and z_3^{*b}.
2. Two clustering assignments, $\psi(z_1^{*b}, k, \lambda)$ and $\psi(z_2^{*b}, k, \lambda)$ are made on z_1^{*b} and z_2^{*b} respectively.
3. $S(\psi, k, \lambda, n)$ is estimated as the distance between $\psi(z_1^{*b}, k, \lambda)$ and $\psi(z_2^{*b}, k, \lambda)$ on z_3^{*b},

$$\hat{S}^{*b}(\psi, k, \lambda, n) = \binom{n}{2}^{-1} |(i,j)_{i<j} : I(\hat{\psi}_1^{*b}(X_i^{(3)}) = \hat{\psi}_1^{*b}(X_j^{(3)})) \neq I(\hat{\psi}_2^{*b}(X_i^{(3)}) = \hat{\psi}_2^{*b}(X_j^{(3)}))| \quad (3.1)$$

   where $\hat{\psi}_1^{*b} = \psi(z_1^{*b}, K, \lambda)$ and $\hat{\psi}_2^{*b} = \psi(z_2^{*b}, K, \lambda)$, $X_i^{(3)}$ and $X_j^{(3)}$ are elements in Z_3^{*b}, and $|A|$ is the cardinality set of A.
4. For each λ, $\hat{k}_\lambda = mode(\hat{k}_\lambda^{*1}, \ldots, \hat{k}_\lambda^{*B})$, where $\hat{k}_\lambda^{*b} = \min_{2 \leq k \leq K} \hat{S}^{*b}(\psi, k, \lambda, n)$
5. Then K is estimated as $\hat{K} = mode(\hat{K}_\lambda)$
6. Given $\hat{K}$, λ is estimated as $\hat{\lambda} = mode(\hat{\lambda}^{*1}, \ldots, \hat{\lambda}^{*B})$, where $\hat{\lambda}^{*b} = \min_\lambda \hat{S}^{*b}(\psi, \hat{K}, \lambda, n)$

Of course, if the number of clusters, K, is known this algorithm can be adapted to use solely for tuning parameter selection. We intend to explore the application of this method towards sparse K–means as well as our Stability Analysis of sparse K–means algorithm below.

More research has been done on the selection of tuning parameters via a stability analysis based on the features selected, in a penalized Least Squares Regression setting [Sun et al., 2013]. We may also try to extract concepts learned from this research and apply it to our clustering algorithm to enhance our results.

Although there is ambiguity to an exact cutoff value for determining consistent selection of a feature amongst the subsamples, our sensitivity analysis has shown that in most applications this decision will not greatly impact the results. We suggest using prediction strength to decide whether to use one or two iterations and for the case where there is a lot of noise, use lenient cutoffs, or the particular cutoff with highest prediction strength. We hope in the future to develop a more systematic approach for determining consistency of feature selection.

3.1 CONCLUSION

As high throughput data is becoming more prevalent, there is an increasing necessity for a stable and accurate clustering method. Equivalently necessary, there is a need to yield a consistent and accurate subset of the features. With Stable Sparse K–means, we have succeeded at providing a reliable method for researchers to investigate and explore their data with. Using simulations and application to two real leukemia data sets, we have demonstrated a method which accomplishes both of these needs simultaneously. In particular, we have shown that in a wide variety of situations, the Stable Sparse K–means method has comparable results, if not better, in terms of clustering accuracy and better results in terms of sparsity and consistency of feature selection.

The R package sskm implementing the method proposed in this dissertation is available on CRAN at https://cran.r-project.org.

APPENDIX A

R CODE- SIMULATIONS

```
library(sparcl)

working.dir<-getwd()
q<-50
p<-1000
n<-20
K<-3
N<-K*n
mu<-0.8
B<-100
pi<-c(0.2,0.3,0.4)
pi2<-c(0.5,0.6,0.7)
filename<-paste(working.dir,"/",p,"_",mu,"
    _thirty_two_simulations_sd.csv",sep="")


# Function to simulate data
sim.samp<-function(q,p,n) {
```

```
dat1<-matrix(rnorm(n*q,-mu,1),n,q)
dat2<-matrix(rnorm(n*(p-q),0,1),n,(p-q))
C1<-cbind(dat1,dat2)
C2<-matrix(rnorm(n*p,0,1),n,p)
dat3<-matrix(rnorm(n*q,mu,1),n,q)
dat4<-matrix(rnorm(n*(p-q),0,1),n,(p-q))
C3<-cbind(dat3,dat4)
data<-rbind(C1,C2,C3)
}

# Function to simulate subsamples from data
sub.sim<-function(data,N,B)  {
m<-N/2
a<-c(1:N)
select<-matrix(NA,nrow=m,ncol=B)
z<-replicate(n=B,expr=list())

for(i in 1:B){

select[,i]<-sample(a,m,replace=F)
z[[i]]<-data[select[,i],]
}
z
}

# Real CER FUNCTION
cer<-function(clusters) {
true.clusters<-c(rep(1,n),rep(2,n),rep(3,n))
tmp<-cbind(dat,true.clusters)
tmp<-cbind(tmp,clusters)
```

```
t<-matrix(NA, nrow=N, ncol=N)
Q<-matrix(NA, nrow=N, ncol=N)

for(i in 1:N){
for(j in 1:N){
if (tmp[i,p+1]==tmp[j,p+1]){
t[i,j]=1
} else {
t[i,j]=0
}
}
}

for(i in 1:N){
for(j in 1:N){
if (tmp[i,p+2]==tmp[j,p+2]){
Q[i,j]=1
} else {
Q[i,j]=0
}
}
}

M=abs(t-Q)
cer<-sum((M/2)/choose(N,2))
cer
}

arg=as.numeric(Sys.getenv("SGE_TASK_ID"))
print(arg)
```

```
seed<-11*arg
set.seed(seed)
dat<-sim.samp(q,p,n)
sub.sample<-sub.sim(dat,N,B)



results<-replicate(n=B,expr=list())
tun_par<-replicate(n=B,expr=list())
wts<-matrix(NA,nrow=p,ncol=B)
all<-matrix(NA,nrow=p,ncol=B)

# Calculate tuning parameters for all subsamples
set.seed(seed)
for(i in 1:B){
print(i)
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample[[i
    ]],K=3), error = function(e) e)
if ('error' %in% class(tun_par[[i]])) {
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample[[i
    ]],K=3), error = function(e) e)
while('error' %in% class(tun_par[[i]]) ) {
refill<-as.data.frame(sub.sim(dat,N,1)  )
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
    error = function(e) e)
sub.sample[[i]]<-refill
}
}
}

tun_sel<-function(tun) {
```

```
a<-as.data.frame(cbind(tun[[6]],tun[[3]],tun[[4]],tun$sdgaps))
names(a)<-c("wbound","nnonzerows","gaps","sdgaps")
max.index <- which.max(a$gaps)
standard <- (a$gaps - a$sdgaps)[max.index]
a$wbound[min(which(a$gaps>=standard))]
}

wb<-lapply(tun_par,tun_sel)

# Perform Sparse K-Means for each Subsample and obtain actual wts
   matrix and positive wts matrix
set.seed(seed)
for (i in 1:B){
results[[i]]<-tryCatch(KMeansSparseCluster(sub.sample[[i]],3,
   wbounds = wb[[i]][[1]], nstart=20,maxiter=6), error= function(e
   ) e)
if('error' %in% class(results[[i]]) ) {
results[[i]] = tryCatch( KMeansSparseCluster(sub.sample[[i]],3,
   wbounds = wb[[i]][[1]], nstart=20,maxiter=6), error = function(
   e) e)
while ('error' %in% class(results[[i]]) ) {

refill<-as.data.frame(sub.sim(dat,N,1) )
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
   error = function(e) e)
if('error' %in% class(tun_par[[i]]) ) {
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
   error = function(e) e)
while ('error' %in% class(tun_par[[i]]) ) {
```

```
refill<-as.data.frame(sub.sim(dat,N,1)   )
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
   error = function(e) e)
}

}
wb[[i]]=tun_sel(tun_par[[i]])
results[[i]] = tryCatch( KMeansSparseCluster(refill,3,wbounds = wb
   [[i]][[1]], nstart=20,maxiter=6), error = function(e) e)
}

}
wts[,i]<-results[[i]][[1]]$ws
all[,i]<-wts[,i]>0
}
all<-all*1


# Obtain matrix of true signal with positive wts
real<-all[c(1:q),]

# Compare to sparse k-means on data
set.seed(seed)
sparse.tun = tryCatch( KMeansSparseCluster.permute(dat,K=3), error
    = function(e) e)
while ('error' %in% class(sparse.tun)) {
sparse.tun = tryCatch( KMeansSparseCluster.permute(dat,K=3), error
    = function(e) e)
}
sparse.wb<-tun_sel(sparse.tun)
```

```
sparse.km<- KMeansSparseCluster(dat,3, wbounds = sparse.wb, nstart
    = 20, silent = T, maxiter=6)
sparse.wts<-sparse.km[[1]]$ws
sparse.all<-sparse.wts>0
sparse.all<-sparse.all*1
sparse.real<-sparse.all[c(1:q)]
sparse.F<-sum(sparse.all)
sparse.F.real<-sum(sparse.real)
sparse.clusters<-sparse.km[[1]]$Cs
sparse.cer<-cer(sparse.clusters)

#Compare to k-means on data
set.seed(seed)
km<-kmeans(dat,3, nstart=20,iter.max=6)
km.clusters<-km$cluster
km.cer<-cer(km.clusters)



#Calculate total number of stable features
for(j in 1:3) {
stab1<-rowSums(all)>=(pi[j]*B)
stab1_real<-rowSums(real)>=(pi[j]*B)
F1<-sum(stab1)
F1_real<-sum(stab1_real)
print(F1)

if (F1==0){
out<-data.frame(seed,p,q,mu,pi[j],F1,F1_real,sparse.F,sparse.F.
   real,sparse.cer,km.cer)
print(out)
```

```
write.table(out,filename,row.names=F,sep=",",append=T)
} else if (F1==1){

# Create new dataset(#2) with only stable features
tmp<-rbind(dat,stab1)
dat2<-tmp[,which(tmp[61,]==1)]
dat2<-dat2[1:(K*n)]

# Run K-Means with reduced dimensions 1st iteration
set.seed(seed)
stable.km<-kmeans(dat2,3, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable1.cer<-cer(stable.km.clusters)

out<-data.frame(seed,p,q,mu,pi[j],F1,F1_real,stable1.cer,sparse.F,
   sparse.F.real,sparse.cer,km.cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
} else {

# Create new dataset(#2) with only stable features
tmp<-rbind(dat,stab1)
dat2<-tmp[,which(tmp[61,]==1)]
dat2<-dat2[1:(K*n),]

# Run K-Means with reduced dimensions 1st iteration
set.seed(seed)
stable.km<-kmeans(dat2,3, nstart=20,iter.max=6)
```

```
stable.km.clusters<-stable.km$cluster
stable1.cer<-cer(stable.km.clusters)

# Take subsamples of new dataset
set.seed(seed)
sub.sample2<-sub.sim(dat2,N,B)



results2<-replicate(n=B,expr=list())
tun_par2<-replicate(n=B,expr=list())
wts2<-matrix(NA,nrow=F1,ncol=B)
all2<-matrix(NA,nrow=F1,ncol=B)

# Calculate tuning parameters for all subsamples of new dataset
set.seed(seed)
for(i in 1:B){
print(i)

tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample2
   [[i]],K=3), error = function(e) e)

if('error' %in% class(tun_par2[[i]]) ) {
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample2
   [[i]],K=3), error = function(e) e)
while ('error' %in% class(tun_par2[[i]]) ) {

refill<-as.data.frame(sub.sim(dat2,N,1)   )
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
   silent=T), error = function(e) e)
sub.sample2[[i]]<-refill
```

```
}

}
}

# Perform Sparse K-Means for each Subsample and obtain actual wts
   matrix and positive wts matrix
set.seed(seed)
wb2<-lapply(tun_par2,tun_sel)
for (i in 1:B){
results2[[i]]<-tryCatch(KMeansSparseCluster(sub.sample2[[i]],3,
   wbounds = wb2[[i]][[1]], nstart=20,maxiter=6), error= function(
   e) e)
if('error' %in% class(results2[[i]]) ) {
results2[[i]] = tryCatch( KMeansSparseCluster(sub.sample2[[i]],3,
   wbounds = wb2[[i]][[1]], nstart=20,maxiter=6), error = function
   (e) e)
while ('error' %in% class(results2[[i]]) ) {

refill<-as.data.frame(sub.sim(dat2,N,1) )
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
    error = function(e) e)
if('error' %in% class(tun_par2[[i]]) ) {
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
    error = function(e) e)
while ('error' %in% class(tun_par2[[i]]) ) {

refill<-as.data.frame(sub.sim(dat2,N,1) )
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
    error = function(e) e)
```

```
}

}
wb2[[i]]=tun_sel(tun_par2[[i]])
results2[[i]] = tryCatch( KMeansSparseCluster(refill,3,wbounds =
    wb2[[i]], nstart=20,maxiter=6), error = function(e) e)
}

}

wts2[,i]<-results2[[i]][[1]]$ws
all2[,i]<-wts2[,i]>0
}
all2<-all2*1

# Obtain matrix of true signal with positive wts
real2<-all2[c(1:F1_real),]

#Calculate total number of stable features
for (k in 1:3) {
stab2<-rowSums(all2)>=(pi2[k]*B)
stab2_real<-rowSums(real2)>=(pi2[k]*B)
F2<-sum(stab2)
F2_real<-sum(stab2_real)
print(F2)

if (F2==0) {
out<-data.frame(seed,p,q,mu,pi[j],pi2[k],F1,F1_real,F2,F2_real,
    stable1.cer,sparse.F,sparse.F.real,sparse.cer,km.cer)
print(out)
```

```
write.table(out,filename,row.names=F,sep=",",append=T)
} else if (F2==1) {

# Create new dataset(#3) with only stable features
tmp<-rbind(dat2,stab2)
dat3<-tmp[,which(tmp[61,]==1)]
dat3<-dat3[1:(K*n)]



# Run K-Means with reduced dimensions 2nd iteration
set.seed(seed)
stable.km<-kmeans(dat3,3, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable2.cer<-cer(stable.km.clusters)

out<-data.frame(seed,p,q,mu,pi[j],pi2[k],F1,F1_real,F2,F2_real,
    stable1.cer,stable2.cer,sparse.F,sparse.F.real,sparse.cer,km.
    cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
} else {

# Create new dataset(#3) with only stable features
tmp<-rbind(dat2,stab2)
dat3<-tmp[,which(tmp[61,]==1)]
dat3<-dat3[1:(K*n),]
```

```
# Run K-Means with reduced dimensions 2nd iteration
set.seed(seed)
stable.km<-kmeans(dat3,3, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable2.cer<-cer(stable.km.clusters)

out<-data.frame(seed,p,q,mu,pi[j],pi2[k],F1,F1_real,F2,F2_real,
    stable1.cer,stable2.cer,sparse.F,sparse.F.real,sparse.cer,km.
    cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
}
}
}
}
```

APPENDIX B

R CODE- LEUKEMIA DATASETS

```
library(sparcl)
library(gplots)
library(fpc)

load("leukemiaS3.Rdata")
working.dir<-getwd()
p<-48788
K<-3
N<-74
B<-100
pi<-c(0.6,0.7,0.8,0.9)
pi2<-c(0.5,0.6,0.7)

filename<-paste(working.dir,"/balgobind.csv",sep="")

verhaak<-S[[2]]
bg.genes<-colnames(verhaak)
```

```
save(bg.genes, file="background_balgobind.rdata")

# Filter out 50% of genes with lowest expression
#verhaak<-as.data.frame(verhaak)
#means<-sapply(verhaak,mean)
#datmeans<-rbind(verhaak,means)
#p50<-as.numeric(quantile(means,0.5))
#high.exp<-datmeans[,datmeans[N+1,]>=p50]

# Filter out low variance
#std<-sapply(high.exp,sd)
#datvar<-rbind(high.exp,std)
#s50<-as.numeric(quantile(std,0.5))
#high.exp.var<-datvar[,datvar[N+2,]>=s50]
#verhaak<-high.exp.var[1:N,]
#p<-dim(verhaak)[2]
ub<-0.7*sqrt(p)

# Function to simulate subsamples from data
sub.sim<-function(data,N,B)  {
m<-round(N/2)
a<-c(1:N)
select<-matrix(NA,nrow=m,ncol=B)
z<-replicate(n=B,expr=list())

for(i in 1:B){

select[,i]<-sample(a,m,replace=F)
z[[i]]<-data[select[,i],]
}
```

```
z
}


# CER FUNCTION
cer<-function(clusters) {
true.clusters<-c(rep(1,27),rep(2,18),rep(3,29))
tmp<-cbind(verhaak,true.clusters)
tmp<-cbind(tmp,clusters)
t<-matrix(NA,nrow=N,ncol=N)
Q<-matrix(NA,nrow=N,ncol=N)

for(i in 1:N){
for(j in 1:N){
if (tmp[i,p+1]==tmp[j,p+1]){
t[i,j]=1
} else {
t[i,j]=0
}
}
}

for(i in 1:N){
for(j in 1:N){
if (tmp[i,p+2]==tmp[j,p+2]){
Q[i,j]=1
} else {
Q[i,j]=0
}
}
}
```

```
}

M=abs(t-Q)
cer<-sum((M/2)/choose(N,2))
cer
}

arg=as.numeric(Sys.getenv("SGE_TASK_ID"))
print(arg)
seed<-11*arg
set.seed(seed)
sub.sample<-sub.sim(verhaak,N,B)


results<-replicate(n=B,expr=list())
tun_par<-replicate(n=B,expr=list())
wts<-matrix(NA,nrow=p,ncol=B)
all<-matrix(NA,nrow=p,ncol=B)

#Function to select predetermined number of genes
sparse_sel<-function(tun) {
a<-as.data.frame(cbind(tun[[6]],tun[[3]],tun[[4]],tun$sdgaps))
names(a)<-c("wbound","nnonzerows","gaps","sdgaps")
a$wbound[a$gaps==max(a$gaps[which(300<=a$nnonzerows & a$nnonzerows
    <=1000)])]
}

#Get list of genes with predetermined number of genes
set.seed(seed)
sparse.pd.tun = tryCatch( KMeansSparseCluster.permute(verhaak,K=3,
```

```
    wbound=seq(2,18,len=10)), error = function(e) e)
while ('error' %in% class(sparse.pd.tun)) {
sparse.pd.tun = tryCatch( KMeansSparseCluster.permute(verhaak,K=3,
    wbound=seq(2,18,len=10)), error = function(e) e)
}
sparse.pd.wb<-sparse_sel(sparse.pd.tun)
sparse.pd.km<- KMeansSparseCluster(verhaak,3, wbounds = sparse.pd.
    wb, nstart = 20, silent = T, maxiter=6)
sparse.pd.wts<-sparse.pd.km[[1]]$ws
sparse.pd.all<-sparse.pd.wts>0
sparse.pd.all<-sparse.pd.all*1
sparse.pd.genes<-sparse.pd.all[sparse.pd.all==1]

#Save list of predetermined number of genes for Pathway Enrichment
    Analysis
sparse.pd.genes<-names(sparse.pd.genes)
sparse.pd.genes.file<-paste("balgobind_sparse_pd_",seed,".rdata",
    sep="")
save(sparse.pd.genes,file=sparse.pd.genes.file)

# Calculate tuning parameters for all subsamples
set.seed(seed)
for(i in 1:B){
print(i)
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample[[i
    ]],K=3,wbounds=seq(1.2,ub,len=10)), error = function(e) e)
if ('error' %in% class(tun_par[[i]])) {
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample[[i
    ]],K=3,wbounds=seq(1.2,ub,len=10)), error = function(e) e)
while('error' %in% class(tun_par[[i]]) ) {
```

```
refill<-as.data.frame(sub.sim(verhaak,N,1)   )
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
   wbounds=seq(1.2,ub,len=10)), error = function(e) e)
sub.sample[[i]]<-refill
}
}
}

tun_sel<-function(tun) {
a<-as.data.frame(cbind(tun[[6]],tun[[3]],tun[[4]],tun$sdgaps))
names(a)<-c("wbound","nnonzerows","gaps","sdgaps")
max.index <- which.max(a$gaps)
standard <- (a$gaps - a$sdgaps)[max.index]
a$wbound[min(which(a$gaps>=standard))]
}



wb<-lapply(tun_par,tun_sel)

#Save Results

wspace=paste("ws",seed,".RData",sep="")
save.image(file=wspace)



# Perform Sparse K-Means for each Subsample and obtain actual wts
   matrix and positive wts matrix
set.seed(seed)
for (i in 1:B){
```

```
results[[i]]<-tryCatch(KMeansSparseCluster(sub.sample[[i]],3,
    wbounds = wb[[i]][[1]], nstart=20,maxiter=6), error= function(e
    ) e)
if('error' %in% class(results[[i]]) ) {
results[[i]] = tryCatch( KMeansSparseCluster(sub.sample[[i]],3,
    wbounds = wb[[i]][[1]], nstart=20,maxiter=6), error = function(
    e) e)
while ('error' %in% class(results[[i]]) ) {

refill<-as.data.frame(sub.sim(verhaak,N,1) )
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
    wbound=seq(1.2,ub,len=10)), error = function(e) e)
if('error' %in% class(tun_par[[i]]) ) {
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3),
    error = function(e) e)
while ('error' %in% class(tun_par[[i]]) ) {

refill<-as.data.frame(sub.sim(verhaak,N,1) )
tun_par[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
    wbound=seq(1.2,ub,len=10)), error = function(e) e)
}

}
wb[[i]]=tun_sel(tun_par[[i]])
results[[i]] = tryCatch( KMeansSparseCluster(refill,3,wbounds = wb
    [[i]][[1]], nstart=20,maxiter=6), error = function(e) e)
}

}
wts[,i]<-results[[i]][[1]]$ws
```

```
all[,i]<-wts[,i]>0
}
all<-all*1


# Compare to sparse k-means on data
set.seed(seed)
sparse.tun = tryCatch( KMeansSparseCluster.permute(verhaak,K=3,
   wbound=seq(1.2,ub,len=10)), error = function(e) e)
while ('error' %in% class(sparse.tun)) {
sparse.tun = tryCatch( KMeansSparseCluster.permute(verhaak,K=3,
   wbound=seq(1.2,ub,len=10)), error = function(e) e)
}
sparse.wb<-tun_sel(sparse.tun)
sparse.km<- KMeansSparseCluster(verhaak,3, wbounds = sparse.wb,
   nstart = 20, silent = T, maxiter=6)
sparse.wts<-sparse.km[[1]]$ws
sparse.all<-sparse.wts>0
sparse.all<-sparse.all*1
sparse.genes<-sparse.all[sparse.all==1]


#Save list of genes for pathway enrichment analysis
sparse.genes<-names(sparse.genes)
sparse.genes.file<-paste("balgobind_sparse_",seed,".rdata",sep="")
save(sparse.genes,file=sparse.genes.file)

sparse.F<-sum(sparse.all)
sparse.F
sparse.clusters<-sparse.km[[1]]$Cs
```

```
sparse.cer<-cer(sparse.clusters)

#Create Heatmap for Sparse k-means results
tmp<-rbind(verhaak,sparse.all)
dat2<-tmp[,which(tmp[(N+1),]==1)]
dat2<-dat2[1:N,]
dat2<-cbind(dat2,sparse.clusters)
dat2<-dat2[order(dat2[,"sparse.clusters"]),]
dat2<-dat2[,1:(ncol(dat2)-1)]
hmdat<-t(dat2)

c1<-rainbow(1,start=0)
c2<-rainbow(1,start=0.5)
c3<-rainbow(1,start=0.9)
cc<-c(rep(c1,sum(sparse.clusters==1)),rep(c2,sum(sparse.clusters
    ==2)),rep(c3,sum(sparse.clusters==3)))

shmfile<-paste(working.dir,"/bb_hm_sparse_",seed,".tif",sep="")
tiff(filename=shmfile)
heatmap.2(hmdat,key=F,dendrogram="row",Colv=F,vline=NULL,hline=
    NULL,col=redgreen,trace="none",ColSideColors=cc)
dev.off()



#Compare to k-means on data
set.seed(seed)
km<-kmeans(verhaak,K, nstart=20,iter.max=6)
km.clusters<-km$cluster
km.cer<-cer(km.clusters)
```

```
#Calculate total number of stable features
for(j in 1:4) {
stab1<-rowSums(all)>=(pi[j]*B)
F1<-sum(stab1)
print(F1)

if (F1==0){
out<-data.frame(seed,pi[j],F1,sparse.F,sparse.cer,km.cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
} else if (F1==1){

# Create new dataset(#2) with only stable features
tmp<-rbind(verhaak,stab1)
dat2<-tmp[,which(tmp[(N+1),]==1)]
dat2<-dat2[1:N,]

#Save list of genes for pathway enrichment analysis
genes1<-colnames(dat2)
genes1.file<-paste("balgobind_genes1_",seed,"_",pi[j],".rdata",sep
   ="")
save(genes1,file=genes1.file)

# Run K-Means with reduced dimensions 1st iteration
set.seed(seed)
stable.km<-kmeans(dat2,K, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable1.cer<-cer(stable.km.clusters)
```

```
#Get heatmap for stable 1st iteration
dat3<-cbind(dat2,stable.km.clusters)
dat3<-dat3[order(dat3[,"stable.km.clusters"]),]
dat3<-dat3[,1:(ncol(dat2)-1)]
hmdat<-t(dat3)

c1<-rainbow(1,start=0)
c2<-rainbow(1,start=0.5)
c3<-rainbow(1,start=0.9)
cc<-c(rep(c1,sum(stable.km.clusters==1)),rep(c2,sum(stable.km.
    clusters==2)),rep(c3,sum(stable.km.clusters==3)))

skhmfile<-paste(working.dir,"/bb_hm_stable_",seed,"_",pi[j],".tif
    ",sep="")
tiff(filename=skhmfile)
heatmap.2(hmdat,key=F,dendrogram="row",Colv=F,vline=NULL,hline=
    NULL,col=redgreen,trace="none",ColSideColors=cc)
dev.off()

out<-data.frame(seed,pi[j],F1,stable1.cer,sparse.F,sparse.cer,km.
    cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
} else {

# Create new dataset(#2) with only stable features
tmp<-rbind(verhaak,stab1)
dat2<-tmp[,which(tmp[(N+1),]==1)]
```

```
dat2<-dat2[1:N,]
ubF<-0.7*sqrt(F1)

#Save list of genes for pathway enrichment analysis
genes1<-colnames(dat2)
genes1.file<-paste("balgobind_genes1_",seed,"_",pi[j],".rdata",sep
    ="")
save(genes1,file=genes1.file)

# Run K-Means with reduced dimensions 1st iteration
set.seed(seed)
stable.km<-kmeans(dat2,K, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable1.cer<-cer(stable.km.clusters)

# Calculate Prediction Strength with reduced dimensions 1st
    iteration
ps<-prediction.strength(dat2,Gmin=3,Gmax=3)
ps1<-ps$mean.pred[[3]]

#Get heatmap for stable 1st iteration
dat3<-cbind(dat2,stable.km.clusters)
dat3<-dat3[order(dat3[,"stable.km.clusters"]),]
dat3<-dat3[,1:(ncol(dat2)-1)]
hmdat<-t(dat3)

c1<-rainbow(1,start=0)
c2<-rainbow(1,start=0.5)
c3<-rainbow(1,start=0.9)
cc<-c(rep(c1,sum(stable.km.clusters==1)),rep(c2,sum(stable.km.
```

```
    clusters==2)),rep(c3,sum(stable.km.clusters==3)))

skhmfile<-paste(working.dir,"/bb_hm_stable_",seed,"_",pi[j],".tif
    ",sep="")
tiff(filename=skhmfile)
heatmap.2(hmdat,key=F,dendrogram="row",Colv=F,vline=NULL,hline=
    NULL,col=redgreen,trace="none",ColSideColors=cc)
dev.off()

# Take subsamples of new dataset
set.seed(seed)
sub.sample2<-sub.sim(dat2,N,B)



results2<-replicate(n=B,expr=list())
tun_par2<-replicate(n=B,expr=list())
wts2<-matrix(NA,nrow=F1,ncol=B)
all2<-matrix(NA,nrow=F1,ncol=B)

# Calculate tuning parameters for all subsamples of new dataset
set.seed(seed)
for(i in 1:B){
print(i)

tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample2
    [[i]],K=3,wbounds=seq(1.2,ubF,len=10)), error = function(e) e)

if('error' %in% class(tun_par2[[i]]) ) {
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(sub.sample2
    [[i]],K=3,wbounds=seq(1.2,ubF,len=10)), error = function(e) e)
```

```
while ('error' %in% class(tun_par2[[i]]) ) {

refill<-as.data.frame(sub.sim(dat2,N,1) )
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
    silent=T,wbounds=seq(1.2,ubF,len=10)), error = function(e) e)
sub.sample2[[i]]<-refill
}

}
}

# Perform Sparse K-Means for each Subsample and obtain actual wts
    matrix and positive wts matrix
set.seed(seed)
wb2<-lapply(tun_par2,tun_sel)
for (i in 1:B){
results2[[i]]<-tryCatch(KMeansSparseCluster(sub.sample2[[i]],K,
    wbounds = wb2[[i]][[1]], nstart=20,maxiter=6), error= function(
    e) e)
if('error' %in% class(results2[[i]]) ) {
results2[[i]] = tryCatch( KMeansSparseCluster(sub.sample2[[i]],K,
    wbounds = wb2[[i]][[1]], nstart=20,maxiter=6), error = function
    (e) e)
while ('error' %in% class(results2[[i]]) ) {

refill<-as.data.frame(sub.sim(dat2,N,1) )
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
    wbounds=seq(1.2,ubF,len=10)), error = function(e) e)
if('error' %in% class(tun_par2[[i]]) ) {
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
```

```
    wbounds=seq(1.2,ubF,len=10)), error = function(e) e)
while ('error' %in% class(tun_par2[[i]]) ) {

refill<-as.data.frame(sub.sim(dat2,N,1)  )
tun_par2[[i]] = tryCatch( KMeansSparseCluster.permute(refill,K=3,
    wbounds=seq(1.2,ubF,len=10)), error = function(e) e)
}

}
wb2[[i]]=tun_sel(tun_par2[[i]])
results2[[i]] = tryCatch( KMeansSparseCluster(refill,K,wbounds =
    wb2[[i]], nstart=20,maxiter=6), error = function(e) e)
}

}

wts2[,i]<-results2[[i]][[1]]$ws
all2[,i]<-wts2[,i]>0
}
all2<-all2*1


#Calculate total number of stable features
for (k in 1:3) {
stab2<-rowSums(all2)>=(pi2[k]*B)
F2<-sum(stab2)
print(F2)

if (F2==0) {
out<-data.frame(seed,pi[j],pi2[k],F1,F2,stable1.cer,sparse.F,
```

```
    sparse.cer,km.cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
} else if (F2==1) {

# Create new dataset(#3) with only stable features
tmp<-rbind(dat2,stab2)
dat3<-tmp[,which(tmp[(N+1),]==1)]
dat3<-dat3[1:N,]

#Save list of genes for pathway enrichment analysis
genes2<-colnames(dat3)
genes2.file<-paste("balgobind_genes2_",seed,"_",pi[j],"_",pi2[k
    ],".rdata",sep="")
save(genes2,file=genes2.file)



# Run K-Means with reduced dimensions 2nd iteration
set.seed(seed)
stable.km<-kmeans(dat3,K, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable2.cer<-cer(stable.km.clusters)

#Get heatmap for stable 2nd iteration
dat4<-cbind(dat3,stable.km.clusters)
dat4<-dat4[order(dat4[,"stable.km.clusters"]),]
dat4<-dat4[,1:(ncol(dat3)-1)]
hmdat<-t(dat4)
```

```
c1<-rainbow(1,start=0)
c2<-rainbow(1,start=0.5)
c3<-rainbow(1,start=0.9)
cc<-c(rep(c1,sum(stable.km.clusters==1)),rep(c2,sum(stable.km.
    clusters==2)),rep(c3,sum(stable.km.clusters==3)))

s2khmfile<-paste(working.dir,"/bb_hm_stable2_",seed,"_",pi[j],"_",
    pi2[k],".tif",sep="")
tiff(filename=s2khmfile)
heatmap.2(hmdat,key=F,dendrogram="row",Colv=F,vline=NULL,hline=
    NULL,col=redgreen,trace="none",ColSideColors=cc)
dev.off()

out<-data.frame(seed,pi[j],pi2[k],F1,F2,stable1.cer,stable2.cer,
    sparse.F,sparse.cer,km.cer)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
} else {

# Create new dataset(#3) with only stable features
tmp<-rbind(dat2,stab2)
dat3<-tmp[,which(tmp[(N+1),]==1)]
dat3<-dat3[1:N,]

#Save genes list for pathway enrichment analysis
genes2<-colnames(dat3)
genes2.file<-paste("balgobind_genes2_",seed,"_",pi[j],"_",pi2[k
    ],".rdata",sep="")
save(genes2,file=genes2.file)
```

```
# Run K-Means with reduced dimensions 2nd iteration
set.seed(seed)
stable.km<-kmeans(dat3,K, nstart=20,iter.max=6)
stable.km.clusters<-stable.km$cluster
stable2.cer<-cer(stable.km.clusters)

# Calculate Prediction Strength with reduced dimensions 1st
    iteration
ps<-prediction.strength(dat3,Gmin=3,Gmax=3)
ps2<-ps$mean.pred[[3]]

#Get heatmap for stable 2nd iteration
dat4<-cbind(dat3,stable.km.clusters)
dat4<-dat4[order(dat4[,"stable.km.clusters"]),]
dat4<-dat4[,1:(ncol(dat3)-1)]
hmdat<-t(dat4)

c1<-rainbow(1,start=0)
c2<-rainbow(1,start=0.5)
c3<-rainbow(1,start=0.9)
cc<-c(rep(c1,sum(stable.km.clusters==1)),rep(c2,sum(stable.km.
    clusters==2)),rep(c3,sum(stable.km.clusters==3)))

s2khmfile<-paste(working.dir,"/bb_hm_stable2_",seed,"_",pi[j],"_",
    pi2[k],".tif",sep="")
tiff(filename=s2khmfile)
heatmap.2(hmdat,key=F,dendrogram="row",Colv=F,vline=NULL,hline=
    NULL,col=redgreen,trace="none",ColSideColors=cc)
```

```
dev.off()

out<-data.frame(seed,pi[j],pi2[k],F1,F2,stable1.cer,stable2.cer,
    sparse.F,sparse.cer,km.cer,ps1,ps2)
print(out)

write.table(out,filename,row.names=F,sep=",",append=T)
}
}
}
}
```

BIBLIOGRAPHY

Martin Azizyan, Aarti Singh, and Larry Wasserman. Feature selection for high-dimensional clustering. 2014.

Brian V Balgobind, Marry M Van den Heuvel-Eibrink, Renee X De Menezes, Dirk Reinhardt, Iris HIM Hollink, Susan TJCM Arentsen-Peters, Elisabeth R van Wering, Gertjan JL Kaspers, Jacqueline Cloos, Evelien SJM de Bont, et al. Evaluation of gene expression signatures predictive of cytogenetic and molecular subtypes of pediatric acute myeloid leukemia. *Haematologica*, 96(2):221–230, 2011.

Wenzhu Bi, George C Tseng, Lisa A Weissfeld, and Julie C Price. Sparse clustering with resampling for subject classification in pet amyloid imaging studies. In *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2011 IEEE*, pages 3108–3111. IEEE, 2011.

Howard D Bondell and Brian J Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with oscar. *Biometrics*, 64(1):115–123, 2008.

Sandrine Dudoit and Jane Fridlyand. A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology*, 3(7):1–21, 2002.

George H Dunteman. *Principal components analysis*, volume 69. Sage, 1989.

Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

Yixin Fang and Junhui Wang. Selection of the number of clusters via the bootstrap method. *Computational Statistics & Data Analysis*, 56(3):468–477, 2012.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Zhiguang Huo, Ying Ding, Silvia Liu, Steffi Oesterreich, and George Tseng. Meta-analytic framework for sparse k-means to identify disease subtypes in multiple transcriptomic studies. *Journal of the American Statistical Association*, 111(513):27–42, 2016.

Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8): 651–666, 2010.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

Eun-Youn Kim, Seon-Young Kim, Daniel Ashlock, and Dougu Nam. Multi-k: accurate classification of microarray subtypes using ensemble k-means clustering. *BMC bioinformatics*, 10(1):260, 2009.

Geoffrey J. McLachlan, RW Bean, and David Peel. A mixture model-based approach to the clustering of microarray expression data. *Bioinformatics*, 18(3):413–422, 2002.

Nicolai Meinshausen and Peter Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010.

Glenn W Milligan and Martha C Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine learning*, 52(1-2):91–118, 2003.

Wei Pan and Xiaotong Shen. Penalized model-based clustering with application to variable selection. *The Journal of Machine Learning Research*, 8:1145–1164, 2007.

Douglas Steinley. Profiling local optima in k-means clustering: Developing a diagnostic technique. *Psychological methods*, 11(2):178, 2006.

Douglas Steinley. Stability analysis in k-means clustering. *British Journal of Mathematical and Statistical Psychology*, 61(2):255–273, 2008.

Douglas Steinley and Michael J Brusco. Choosing the number of clusters in k-means clustering. *Psychological methods*, 16(3):285, 2011.

Wei Sun, Junhui Wang, Yixin Fang, et al. Regularized k-means clustering of high-dimensional data and its asymptotic consistency. *Electronic Journal of Statistics*, 6:148–167, 2012.

Wei Sun, Junhui Wang, and Yixin Fang. Consistent selection of tuning parameters via variable selection stability. *The Journal of Machine Learning Research*, 14(1):3419–3440, 2013.

Stephen Swift, Allan Tucker, Veronica Vinciotti, Nigel Martin, Christine Orengo, Xiaohui Liu, and Paul Kellam. Consensus clustering and functional interpretation of gene-expression data. *Genome biology*, 5(11):R94, 2004.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Robert Tibshirani and Guenther Walther. Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14(3):511–528, 2005.

Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

George C Tseng and Wing H Wong. Tight clustering: A resampling-based approach for identifying stable and tight patterns in data. *Biometrics*, 61(1):10–16, 2005.

William Valdar, Christopher C Holmes, Richard Mott, and Jonathan Flint. Mapping in structured populations by resample model averaging. *Genetics*, 182(4):1263–1277, 2009.

Roel GW Verhaak, Bas J Wouters, Claudia AJ Erpelinck, Saman Abbas, H Berna Beverloo, Sanne Lugthart, Bob Löwenberg, Ruud Delwel, and Peter JM Valk. Prediction of molecular subtypes in acute myeloid leukemia based on gene expression profiling. *haematologica*, 94 (1):131–134, 2009.

Junhui Wang. Consistent selection of the number of clusters via crossvalidation. *Biometrika*, 97(4):893–904, 2010.

Daniela M Witten and Robert Tibshirani. A framework for feature selection in clustering. *Journal of the American Statistical Association*, 2012.

Benhuai Xie, Wei Pan, Xiaotong Shen, et al. Penalized model-based clustering with cluster-specific diagonal covariance matrices and grouped variables. *Electronic Journal of Statistics*, 2:168–212, 2008.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.